2007:138 CIV

MASTER'S THESIS

Evaluation of European SRAM-based FPGA Using the ESA VHDL IP-Core Library

HÅKAN HELZENIUS

MASTER OF SCIENCE PROGRAMME
in Space Engineering

Luleå University of Technology
Department of Space Science, Kiruna

LULEÅ UNIVERSITY OF TECHNOLOGY

2007:138 CIV • ISSN: 1402 - 1617 • ISRN: LTU - EX - - 07/138 - - SE

Evaluation of a European SRAM-based FPGA using the ESA VHDL IP-Core library

Håkan Helzenius

Master of Science Programme in Space Engineering:
Department of Space Science

Luleå University of Technology

Gävle, March 2007

ABSTRACT

AT40KEL-DK is a design kit from ATMEL, including the Rad Hard SRAM-based reprogrammable FPGA AT40KEL040. The use of reprogrammable FPGAs in space is fairly limited since they are often sensitive to radiation. Today only a few reprogrammable FPGAs have flown on real space missions, but in the near future this technology can cut down on development time and open up for new applications in space. Most of the mentioned technology is at present developed in USA with export restrictions, which makes it even more interesting with a European product.

This Master Thesis has focused on the overall evaluation of AT40KEL-DK and all of its content. The design kit contains both the evaluation board and the FPGA, together with the required software. The approach has been to use the ESA VHDL IP-core library and try to implement one of its designs into the FPGA. The IP-core chosen was a SpaceWire codec developed by the University of Dundee. In addition to the test with the codec, the FPGA was also tested with smaller designs to assess additional characteristics like clock speed limitations.

The timeframe of the project was rather limiting and there was only time for a few test results. Instead the project focused on its main objective, a general assessment of the whole design kit. The experiences of both the software and the hardware were documented and given as feedback to ATMEL, as well as laying the foundation for further testing by ESA.

SAMMANFATTNING

AT40KEL-DK är en utvecklingssats från ATMEL, vilket inkluderar den strålnings-härdade SRAM-baserade omprogrammerbara FPGA:n, AT40KEL040. Användandet av omprogrammerbara FPGA:er i rymden är relativt begränsat eftersom de ofta är känsliga för strålning. Idag så har endast några få omprogrammerbara FPGA:er använts i rymden, men inom en snar framtid kan denna teknologi både skära ned på utvecklingskostnader samt öppna upp för nya användningsområden i rymden. Nästan all den involverade teknologin är i dag utvecklad i USA med export restriktioner, därför är det extra intressant med en Europeisk produkt.

Det här examensarbetet har fokuserat på att utvärdera helheten av AT40KEL-DK med all dess innehåll. Utvecklingssatsen innehåller både ett kretskort för att testa FPGA:n samt all mjukvara som behövs. Infallsvinkeln har varit att använda ESA:s VHDL IP-bibliotek och försöka att implementera en av dess konstruktioner i FPGA:n. Från biblioteket valdes en SpaceWire-codec utvecklad av Universitetet i Dundee. Utöver testet med SpaceWire så var FPGA:n också testad med mindre konstruktioner för att utvärdera andra aspekter, som t.ex. den maximala klockhastigheten under varierande omständigheter.

Tiden begränsade dock projektet kraftigt och det fanns bara tid för några få testresultat. Istället så fokuserade projektet på dess huvudmål, en generell utvärdering av hela utvecklingssatsen. Erfarenheterna från både mjukvara och hårdvara var dokumenterade och gavs sedan som feedback till ATMEL, men har samtidigt skapat en bra grund för fortsatt testning av ESA.

PREFACE

This thesis is the final report of the Master of Science programme in Space Engineering at Luleå University of Technology. The work has been carried out in the Microelectronics Section at the European Space Research and Technology Centre, ESTEC, one of European Space Agency's key establishments, situated in Noordwijk, the Netherlands.

The work has been very interesting and rewarding and I will look back with fond memories on my time at ESTEC. I would like to thank the Microelectronics Section and especially my supervisor David Merodio Codinachs for the warm welcome and support. Thanks also to the section head Agustín Fernández León for letting me come and be a guest in their section for these month. I would like to thank the Payload Section as well, for making their lab and equipment available to me. Last but not least, thanks also to all new friends that have made this time truly wonderful.

Håkan Helzenius
Gävle, March 2007

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

Microelectronics have at all times been a complicated area in space technology, the radiation environment have always created heavy restriction. This is still true today, but development of new and better devices has steadily opened up this field for new applications. FPGAs have been used in space for some time now, but that is mostly One-Time Programmable devices. Only a few reprogrammable FPGA have flown on real space missions, but in the near future this technology can cut down on development time and open up for new exciting applications in space. The Microelectronics Section in ESA has received a design kit, AT40KEL-DK, for a reprogrammable FPGA developed in Europe by ATMEL. This FPGA is radiation harded and is of large interest for ESA and parts of the space industry.

This report has focused on the overall evaluation of AT40KEL-DK and all of its content. The design kit contains both the evaluation board and the FPGA, together with the required software. The approach has been to use the ESA VHDL IP-core library and try to implement one of its designs into the FPGA. The IP-core chosen was a SpaceWire codec developed by the University of Dundee. In addition to the test with the codec, the FPGA was also tested with smaller designs to assess additional characteristics like clock speed limitations.

The report will first give a brief background of subjects touched by the project and then describe the project itself and its objectives. It is then possible to read about the preparations before the description of the two major tests are presented in Ch. 6 and 7. The report is then to be completed with a conclusion and discussion.

2 GLOSSARY

DSP	Digital Signal Processing
DUT	Device Under Test
ESA	European Space Agency
FPGA	Field Programmeble Gate Array
FIFO	First In First Out
HDL	Hardware Description Language
IP	Intellectual Product
ITAR	International Traffic in Arms Regulations
LB	Logic Block
LUT	Lookup Table
LVDS	Low Voltage Differential Signaling
OTP	One-Time Programmable
RAM	Random Access Memory
SEU	Single Event Upset
SpW	SpaceWire
SRAM	Static Random Access Memory
TID	Total-Ionizing Dose
TMR	Triple Modular Redundancy
UART	Universal Asynchronous Receiver-Transmitter
VHDL	Very-High-Speed Integrated Circuit HDL

3 BACKGROUND

3.1 FPGA

FPGA stand for Field-Programmable Gate Array and is an integrated circuit that can be programmed after it is manufactured, hence its name "Field Programmable". There are two different types of FPGAs, one that only can be programmed once called One-Time Programmable, OTP, and one that is reprogrammable. OTP technology is much more common in space qualified designs because it is in general more robust than the reprogrammable one. But as Bonacini et al. (2006) writes, the in-system reprogrammability of FPGAs is of great importance giving extreme flexibility to the application, which can be updated in case of changing requirements or failure recovery. The issue is the reprogrammable FPGA's sensitivity to radiation, which will be discussed more in detail in the next chapter.

To be programmable a FPGA contains numerous configuration switches that after programming routes the different Logic Blocks, LB, together. Described by Pellerin and Thibault (2005), a typical FPGA contains Logic Blocks that make up the bulk of the device and they are based on Lookup Tables, LUT, (of perhaps four or five binary inputs) combined with one or two single-bit registers and additional logic elements such as clock enables and multiplexers. These Logic Blocks and LUTs look differently depending on the technology used, different companies use different technologies, but it usually also differ between product series within a company. These Logic Blocks are then connected through a grid surrounding them, which also connect with the I/O pins at the edges of the chip. See figure 3-1.

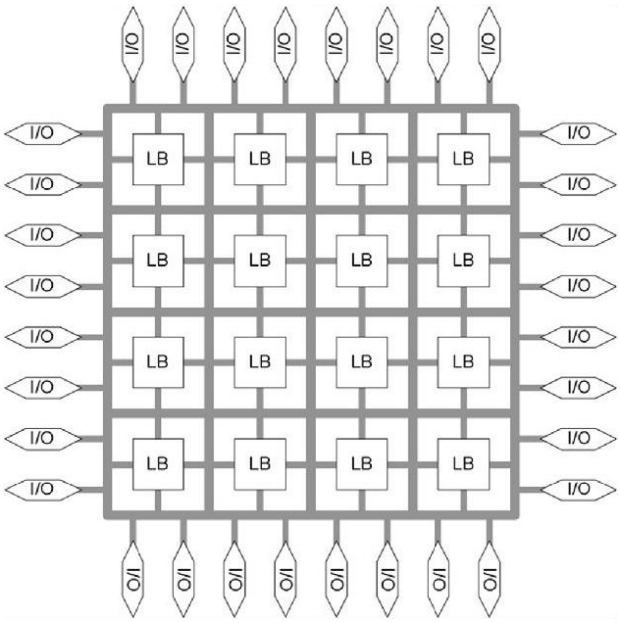

Figure 3-1. FPGA structure

Many FPGAs provide internal SRAMs located between the Logic Blocks to greatly improve functionality. This feature opens up to new possibilities although it also brings about another trait that is usually sensitive to radiation. But despite that the use of SRAM-based FPGA is growing in space based applications because of low application development cost, short time to market, and the reprogramming flexibility that they offer, as Tiwari and Tomko (2005) describes. The need is recognised and today there are FPGAs with radiation tolerant SRAM designs. Other FPGAs also include additional internal blocks to increase the performance in different applications: like internal DLLs/PLLs; multipliers or even more advanced DSP block (DSP Slices, …); hard-macro processors (PowerPCs, …); high speed serial links; etc. But these bring about even more radiation issues to overcome.

3.1.1 FPGAS AND RADIATION

As have been mentioned in the previous chapter there are several issues with FPGAs and radiation. Many studies have been done on the radiation effects on FPGAs and proved them to be often sensitive to both Total-Ionizing Dose, TID, and Single-Event Upsets, SEUs, (Bonacini et al. 2006). These two radiation effects are very different and the problems need to be solved in different ways. They differ that much because of their different sources. As Tiwari and Tomko (2005) explain, TID is caused mainly by composite of gamma rays, x-rays, and other radiation particles. Ion radiation, low energy alpha particles, and neutron radiation are responsible for single event effects. SEU is a common type of single event effect and implies that a bit-flip has occurred somewhere in the device. A bit-flip in an FPGA can also be a more serious concern than in a normal case since it can happen in the configuration logic, if that is the case the device can be rendered defective until it is reprogrammed. In space missions this is a major problem as it can be a huge risk to reprogram a space borne FPGA without the right preparation.

The SEU problem needs to be mitigated in some way and as Sterpone and Violante (2005) states Triple Module Redundancy, TMR, is a known solution for hardening digital logic against SEUs that is widely adopted for traditional techniques. This method or a similar one is most commonly used for this purpose. In its simplest explanation it implements three flip-flops instead of one and then there is always a voting between them. That means that if one flip-flop gets a bit-flip the other two will win the vote and the correct answer will be sent, see figure 3-2.

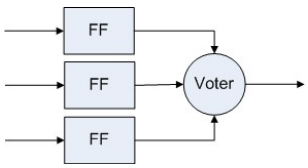

Figure 3-2. TMR

The question is if this method should be applied to every flip-flop and how will it be implemented in the best way. The area the flip-flops occupy will at least be tripled and how is this best designed. Today's Radiation Hard FPGAs intended for space applications have already TMR, or similar, built into the device so that the designer do not have to include it in its design.

3.2 VHDL

VHDL is a Hardware Description Language, HDL, and stands for Very-High-Speed Integrated Circuit HDL. As Balch (2003) writes, Hardware description languages were developed to ease the implementation of large digital designs by representing logic as Boolean equations as well as through the use of higher-level semantic constructs found in mainstream computer programming languages. In this way complex logic can be described in an easier way and with today's modern tools implemented in devices like FPGAs. There are currently two major HDLs in the world, VHDL and Verilog, they have about equally many devoted users around the globe.

VHDL was the original and first hardware description language to be standardized by the Institute of Electrical and Electronics Engineers, through the IEEE 1076 standard (Pedroni 2004). It has since then evolved through standard updates a couple of times. Because it is standardised it is to a great extent independent from various software platforms. As the same design can be simulated and synthesised with tools from different vendors it is not dependent on specific companies. That makes it easier to develop designs or Intellectual Products, IP, now that they can be sold or licensed more independently to others. There is more information on how to implement a design into a device, in the chapter "Project Design Flow".

Instead of drawing schematics the design can now be described in code. This may not be very helpful with very small designs, but when it becomes a little bit larger it will be of immensely more help. There are roughly three levels of coding in VHDL, there is Gate Level, Register Transfer Level and Behavioral Level. It is possible to use all three levels in the same code, but it is good to keep them in mind when planning the design and not to mix them so that it will be hard to follow the code. It is also important to remember that all code that can be simulated does not have to be synthesizable. VHDL-code can look in many different ways even though it has a common structure, the example below show a very basic design in VHDL.

Example 3-1:

```
-------------------------------------------------------------------------------------------------
Library ieee;                                        -- Adding libraries
  use ieee.std_logic_1164.all;

Entity adder is                                      -- Port declaration
  port( x,y,c_in:  in     std_logic;
        s,c_ut:    out    std_logic);
end;

Architecture behv of adder is                        -- Structure and behaviour
begin
                                                     -- Gate level coding
  s   <= (x XOR y) XOR c_in;
  c_ut <= (x AND y) OR (x AND c_in) OR (y AND c_in);

end;
-------------------------------------------------------------------------------------------------
```

3.3 ESA IP-cores

The Microelectronics section at ESA administrates many different IP-cores. These have been developed in the scope of ESA activities, both as in-house developments and as contracting work. The complexity range from smaller designs to larger System On Chip devices (Microelectronic Section's webpage, 2007). The IP-cores can also be licensed from ESA with special restrictions; more information about the IP-cores can be found at the Microelectronic Section's webpage:

http://microelectronics.esa.int/core/corepage.html

3.4 SpaceWire

SpaceWire, SpW, is a standard for inter-satellite communication and is described in the document ECSS-E-50-12A, issued by European Cooperation for Space Standardization, ECSS. The SpW technology provide a high speed data link intended to meet the needs for remote sensing instruments and other high demanding space applications. SpaceWire is a full-duplex, bidirectional, serial, point-to-point data link. It encodes data using two differential signal pairs in each direction. That is a total of eight signal wires, four in each direction (ECSS-E-50-12A, 2003). A SpW connection range from 2 to 400 Mb/s, but an updated version, with SpaceFiber as working name, is currently in development and will be able to reach much higher speeds.

The standard describes the technology in several different levels, physical, signal, character, exchange, packet and network level. For information on SpW see the standard, this can be retrieved for a free registration at ECSS's website www.ecss.nl.

3.5 AT40KEL-DK

AT40KEL-DK is a design kit developed by ATMEL to allow designers to evaluate and prototype applications using their rad-hard FPGA, AT40KEL040. The kit includes a motherboard with a configuration EEPROM, a daughterboard with the FPGA already mounted, a standard parallel cable to program the FPGA and CD-ROMs containing software and documentation (AT40KEL-DK Design Kit User Guide, 2006). AT40KEL040 is a relatively small FPGA with around 40000 logic gates, but its rad-hard feature together with its reprogrammability and internal SRAMs makes it an interesting device. Its development has been centred in Nantes, France, which makes it a European product. This means that it is not ITAR restricted and is easier to buy. ITAR is a restriction U.S. government has but on export of advanced military or space technology. Without ITAR the purchaser do not have to state the precise use of the device and ask U.S. government for permission before it can be bought.

The FPGA has not yet been flown on a real space mission but its use is growing and it is likely that it will be flown relatively soon. For the moment Kongsberg Defence & Aerospace in Norway is working on designs that make use of this FPGA. There is also a new generation of this device in development that is larger but built on the same structure.

4 PROJECT DESCRIPTION

4.1 Objectives

The main objective of this project is to make a general assessment of the design kit AT40KEL-DK from ATMEL and the included FPGA, AT40KEL040. It is not only the hardware that is important to evaluate, it is also necessary to test the effectiveness and user friendliness of the belonging software as well.

As a first stage the FPGA will be tested by implementing a SpaceWire-codec from University of Dundee, which will raise the level of experience with the kit and in that way simplify further testing. Problems that might have been overlooked otherwise will emerge before the final testing plan is made for the additional tests. The goal is not to have a fully operational SpW-node working at high speed, but to have basic functionality at low speed.

Different generated macros will also be created and stressed in several ways. This will both examine some of the specific technologies and test the macro generator software. The results are a good indicator of the performance and functionality of the design kit as well. A comparison can then also be made between identical functionalities not created by the macro generator.

4.2 Project Plan

Since the objectives are split up into two major goals, the project plan and even the whole project is divided into two parts. In the first part the goal is to implement a SpW-node into the FPGA and to reach this goal the process has been broken up into several steps.

- The first step is to get a general knowledge of the hardware and software provided by ATMEL, mainly through reading documentation and following tutorials.

- After understanding the essential elements in the implementation process it is important to try the whole design flow with a very simple design. This will give an early indication of possible problems in any of the design stages, so that they can be mitigated before too much work already has been completed.

- Before the SpW-node can be implemented the SpW technology and IP-core must be understood. This will be done by reading the SpW standard and the documentation of the IP-core. Test simulation will also help to get a better understanding.

- The next step is to make the required modifications of the IP-core so that it is compatible with the AT40KEL specific technology. With the adjustments it should be possible to take the design through both synthesis and place & route.

- The design is then downloaded into the FPGA and tested on the prototyping board. At first it will be tested in very low speed, which can in turn be raised when everything is working satisfactory. During this process it is expected that further modification has to be done so this step and the previous one will function as an iterative loop until good results are reached.

- As an optional last step the whole design will be tested together in a “real” SpW network if the possibility is given. The node will not be working in high speed but if it works in initialisation speed it will be considered a success.

The second part of this project is more of a traditional testing structure. Several small designs have been chosen to be implemented into the device and stressed, the results will then be documented and compared. In this way different fundamental designs’ performances can be measured and at the same time a clearer picture is produced around the generated macros. The macros will be tested in two different ways, one where an internal Test Interface is used and one where the macro is connected directly to the pins and probed. The test plan can be seen below and tests within brackets are of lower priority than the others.

	8 bit	16 bit	32 bit	
Only Registers	X			ALL
Ripple Carry Adder				
No Registers	X			
IO Registers	X	X	(X)	
Multiplier				
Signed	X	X	(X)	
Unsigned	X	(X)		
Signed pipeline X1	X	(X)		
Memory				
RAM - Dual Port				
Area Optimised	X	X		
Speed Optimised	X	X		

Table 4-1. Test Plan

First of all the memories will be tested, because the Test Interface need to use buffers and then it is important to now their limitations. The integrated SRAMs are also a special feature that AT40KEL provides and that makes it even more interesting to test. But the memories will only be tested by the Test Interface, as it would be too complicated to control it directly from the pins. The Test Interface will allow for an easier use of larger amount of data as well as providing a better user interface.

After that other designs will be tested, the adder and multiplier designs are chosen because they are so frequently used in almost every design, which makes them of great interest. The test procedure has been divided into the following steps.

- Create a test interface that can communicate with a computer and perform test by instructions. Test that it works properly.

- Start testing the internal SRAMs inside the FPGA, using the Test Interface. Figure 4-1 shows a test setup with the Test Interface.

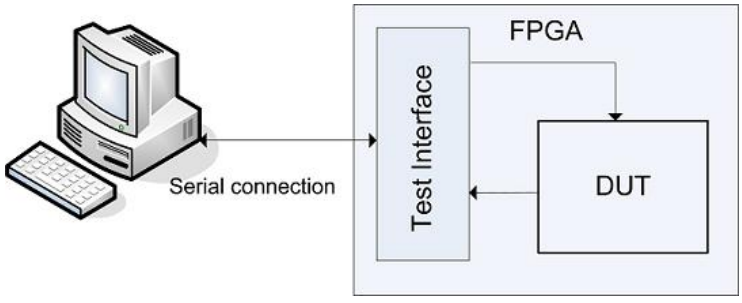

Figure 4-1. Test Setup

- Test the other designs with high priority in the test plan.

- Test the same designs but without the Test Interface and compare with previous results.
- If the time allows, test the lower priority designs.
- As an optional test, designs created without the Macro Generator can be tested and the results compared with the equivalent design already tested.

4.2.1 PROJECT DESIGN FLOW

Figure 4-2 shows a flow chart over the design process when a design is implemented into an FPGA. This flow was followed with all designs in this project and so became an important structure in the project plan. For those that are not familiar with these concepts a short description will follow. For more case specific information see appendix II, Design Flow Guide.

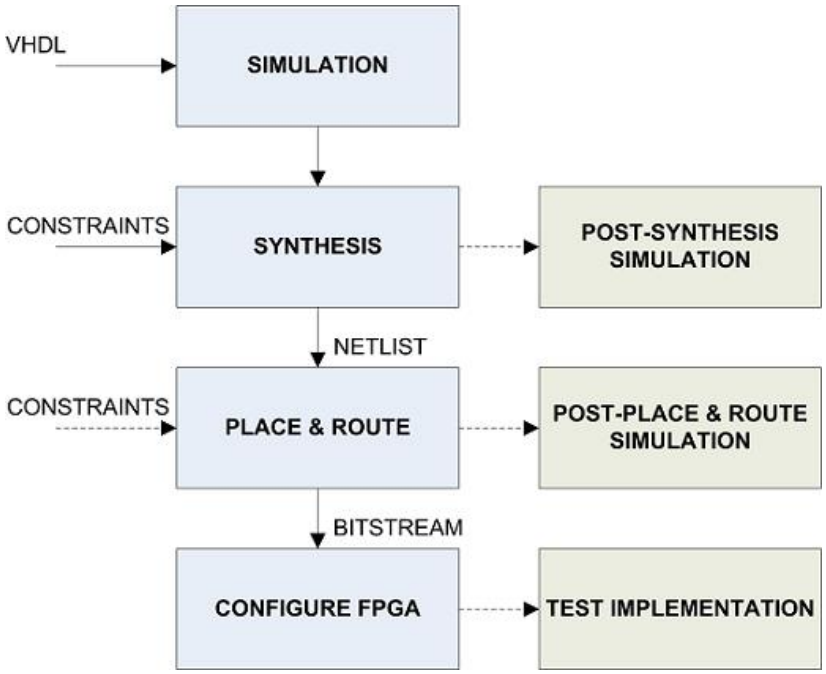

Figure 4-2. Design Flow

First of all the design is written in a Hardware Description Language, HDL, in this project it has always been in VHDL. The design is then run in a simulation program to do early testing and check the correct functionality. After probably a couple of iterations between code writing and simulation a design is ready for synthesis.

During synthesis the code is transformed into a representation of real electronic parts, and on the bottom line logic gates. It is in this process most of the major constraints are put on the design, the most notable is clock speed and input/output delays. During this process it is not unusual with new problems arriving, which needs to be fixed back at the coding stage. These new issues are found during post-synthesis simulation.

Since it is very common with new problems at the first synthesis run, it is important to do the post-synthesis simulation. Without this simulation the problems will not surface until later in the design flow and there is still the need to go back to coding and start over. The simulation makes sure that every part has been connected correctly and the design still does what it is intended to do.

If the simulation is working it is time to import the netlist to the place & route tool. The netlist is a file created by synthesis, containing information about the parts included. During place & route the parts defined by synthesis get their placement inside the FPGA and is then routed together. Now an exact representation, including detailed timing, on how the design will be implemented in the FPGA is created, even though it is not loaded into the FPGA yet. First it is recommended to do a post-place & route simulation to confirm that everything is working properly.

The post-place and route simulation is the last simulation needed, if it works then it is likely that the design will work inside the FPGA itself. This simulation is more accurate since it is able to use timings during simulation. That is possible because signal path length is now known after the routing. With a working simulation it is time to load the design into the FPGA.

The place & route tool gives a bitstream file as a result when it finish. This file is then sent to the FPGA by special software and the FPGA should then be operational. Even if it is likely the design is working it is important to test that everything is functional.

4.3 Deviations during Execution

Unfortunately the timeframe of the project did not allow for the whole project plan to be carried out. In the end the SpaceWire codec never got as far as being implemented into FPGA and the performance tests did not produce as many results as hoped. But interesting results were acquired and plenty of feedback about the overall assessment could be reported.

5 PREPARATION

To begin with, all equipment that could be of use in the lab was tested. If something important was not available or not working the problem was either solved by lending or ordering equipment. It was then verified that the Development Kit contained all of its parts. The board was powered up to see that it did not fail to do so. The software belonging to the Development Kit (System Designer 3.0) was installed at the working station. The attached patches were also added. Modelsim and other useful software were installed as well.

The documentation for the FPGA and its development kit must be read and understood along with documentation for many of the software used. Time was spent getting acquainted with the programs mainly through tutorials and reading.

5.1 Test Environment

All the software oriented tests and developments were made in an office environment on a PC. The computer was running Windows XP and had a Pentium 4 processor running at 2.80GHz with 1GB of RAM. The software used for simulation was Modelsim 6.1b with licence from the Microelectronics section and not the Modelsim provided by System Designer. For synthesis and place & route programs and licences from the development kit were used. These programs were Leonardo Spectrum for Atmel version 2002c.37_OEM_Atmel from Mentor Graphics and IDS Figaro 7.6.7 patch level 3a2 from ATMEL. It is important to point out that System Designer by itself was never used only some of the software that came along with it.

For the hardware tests TEC-EDP's lab was used. Access was granted to computer PC9 for configuring the FPGA while a Logic Analyser and Pattern Generator were also made available. PC9 was using Windows XP and had a 1.70 GHz processor and 512 MB of RAM. The Logic Analyser was an Agilent Technologies 1681 AD while the Pattern Generator was the pattern generator part of TLA 7012 Logic Analyser (portable mainframe) from Tektronix. As a test board for the FPGA the board in the development kit were used. At a very late stage of the project the Logic Analyzer part of TLA 7012 arrived, which proved to be useful for some late testing.

Tools & Equipment list	Description
Equipment	
Work Station, Office PC	XP, Pentium 4, 2.80GHz, 1GB RAM
Work Station, Lab PC (PC9)	XP, 1.70 GHz, 512 MB RAM
Motherboard, Design Kit	ATDH40M from ATMEL
Daughterboard, Design Kit	MQFP160 from ATMEL
Rad Hard FPGA, Design Kit	Part Number: AT40KEL040KW1-E
Parallel cable, Design Kit	Standard

UART adapter/converter	
Serial cable	Standard, Male-Female
Logic Analyser	Agilent Technologies 1681 AD
Pattern Generator/Logic Analyser	Tektronix TLA 7012 Logic Analyser (portable)
Software	
Modelsim SE 6.1b	From ESA lincense
Leonardo Spectrum v2002c.37_OEM_Atmel	From System Designer 3.0
IDS Figaro 7.6.7 patch level 3a2	From System Designer 3.0
CPS 8.05	From System Designer 3.0
Terminal v1.9b	Freeware
Free Hex Editor	Freeware

Table 5-1. Tools and Equipment list

5.2 Testing a small design

After reading the documentation the whole design flow was tested with a very simple design to verify that everything was working properly. In this case both an AND-gate and a counter were taken through the design flow. This process was more time consuming than first thought since the software were made for the commercial FPGAs and when working with the RAD-Hard FPGA many special cases had to be taken into account. First of all System Designer could not be used by itself and instead the programs included in System Designer had to be used on their own. There were also some problems of finding the right library for IDS Figaro but after all the issues were taken care of, all of the equipment worked in the end. For future guidance a simple design flow guide is included as an appendix in this report. Below you can find a print screen from the Logic Analyser demonstrating a working counter. It was an 8 bit counter with an asynchronous reset counting clock pulses.

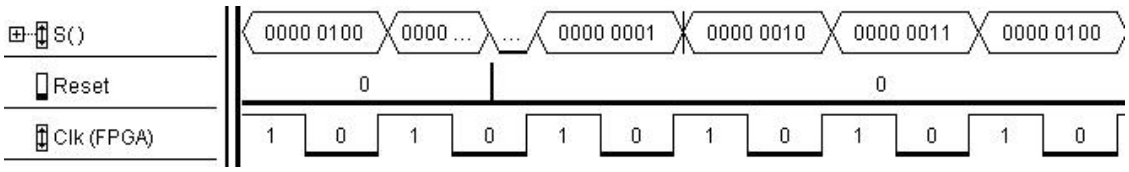

Figure 5-1. Test Signals with a counter

6 THE SPACEWIRE CODEC

6.1 Additional Preparations

For this special test, with the SpaceWire node from University of Dundee, some extra preparation had to be made. The major preparation and the most obvious one were to learn about the SpW technology and this IP-core in particular. The first step was to read the SpW standard, ECSS-E-50-12A, together with some drafts on possible additions to the standard. After getting a general knowledge about SpW the second step was reading the documentation for the IP-core, the two most important documents were the User Manual and the VHDL Functional Description. But also other documents like the RTL Verification User Manual became useful during the process. Except the VHDL Fuctional Description all these documents can be found on internet, the standard can be found on ECSS website while all the others can be found on the Microelectronic Section's website. See the references for more information.

6.2 Execution

At first the SpaceWire IP-core had to be modified to suit the ATMEL technology. The modifications can be divided into to levels of adjustments, one that uses the configuration option already made available by the IP-core and one where code is added or even changed. In the end no code in the IP itself was modified and all customisation were done in wrappers and in added parts. The IP did not include a transceiver FIFO and receiver buffer and instead gave an example on code of how it could look like. This example was important for verification since it was used during the verification test that was delivered together with the IP-core. This example code was not officially a part of the IP but worked as a groundwork on which the modifications were done. These were also the areas most sensitive to changes in technology and could in many cases expect modifications.

The configuration settings were the first to be adjusted because these would be the platform all other changes had to be done. By choosing only one configuration and excluding the others from modification limits the options in the future, but this was deemed reasonable since this project only was supposed to demonstrate a working SpW node in the FPGA and not design a working copy for future projects. At first this setting was chosen:

Bit-clock configuration:

CFG_BITCLK : CfgBitclk_T := SYS_DEFAULT

In this setting only one clock were used, the SYS_DEFAULT option uses the system clock (SYSCLK) also in the transceiver. This option was chosen out of simplicity since it would avoid any problems caused by the use of multiple clock domains.

Unfortunately there were problems with the verification of this setting so after some consideration the setting was changed to TXCLK_DEFAULT. This setting uses a separate clock asynchronous to the system clock for the transceiver. This is also a simple setting and it is used as default setting in the verification test. After the changes the final settings were:

Address length for receiver buffer:
CFG_RXBUF_ADDRLEN : INTEGER = 5 ($5^2 = 32$)
Number of bits used for transmit and slow rate:
CFG_RATE_NUMBITS : INTEGER := 6
Bit-clock configuration:
CFG_BITCLK : CfgBitclk_T := TXCLK_DEFAULT
Set if double data rate should be used:
CFG_DDROUT : STD_LOGIC := '0' (Single Data Rate is used)
Set to one if pipelining is used:
CFG_PIPELINE : STD_LOGIC := '1' (Pipelining is used)
Choose which clock is used to read from read buffer:
CFG_SYNCRDCLK : STD_LOGIC := '1' (Sysclk is chosen)
Set if receiver discards empty packets:
CFG_DISCARD_EMPTY_PKT : STD_LOGIC := '1' (Yes)
Set which slow clock enable, internal or external:
CFG_SLOW_CE_SEL : STD_LOGIC := '1' (External)

After finalising the setting above, work on the transceiver FIFO and the receiver buffer was started. No major changes were needed in this part. The SpW IP used RAM memories with nine data I/O-ports while the macro generator in the place & route tool only could handle a multiple of four ports. This problem was easily solved by using only nine ports out of twelve defined.

A relatively big issue with the FPGA technology was that the internal memory blocks had only one clock input. The FIFO and the Receiver Buffer were designed for a read and a write clock and in that way be able to work as an interface between different clock domains. This problem had to be worked around somehow, it is not impossible, but it would have been convenient to use the memory blocks as interfaces.

After the design modifications the verification test for the node was not longer applicable. It might have been possible to adapt the verification test but it would have taken a great deal of time. Designing a smaller and simpler test for the basic functionality was deemed quicker. The test was only a simple loop back where input and output data were compared at the “backside” of the node.

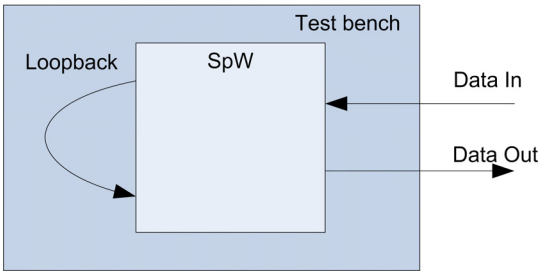

Figure 6-1. Loopback Test

A way of comparing the "data in" with the "data out" automatically, as well as a more advanced test bench were planed but were postponed because of more important task at hand at the time.

The design worked in pre-synthesis simulation but after synthesis it was very hard to verify the design because a convenient way to do this simulation was never found. Integrated software generated macros had to have there code modified by hand to be compatible. In this case it was certain that the design did not work since it encountered problems in the place & route. Configuration signals in the SpW Ip-core could not be handled correctly by the synthesis tool, or at least the place and route tool could not understand the synthesis tool at all time. This was not expected since the IP-core was synthesised by the same tool, Leonardo Spectrum, during development.

6.3 Results

The hopes of having a SpaceWire node working in the FPGA were never fulfilled and therefore no test results from a working version in the lab can be showed. But what is more important is that the experiences of the process can be presented. The main goal of the project was to test the FPGA and the software belonging to it. This objective is accomplished thanks to the extensive work put into solving the numerous issues. Most if the issues were solved especially in collaboration with a fast and helpful support service provided by ATMEL. The following text describes problems encountered and solutions to some of them in addition to suggestions on improvements. Not all problems described are related to the FPGA and its software, even issues not related to the main objective are accounted for here.

Something that early was identified as a difficulty was the single clock input to the embedded RAMs in the FPGA. The SpW node in its core did not need dual port asynchronous RAMs to work, but the receiver buffer and transceiver FIFO would enjoy a big advantage with such an option. The design would have a great benefit of using the RAMs as clock domain interfaces since it is not a trivial problem to get around otherwise.

Initially a lot of problems with the software, System Designer, were encountered, this could be expected with the lack of previous experience in these typical programs but

the learning process could have been made much easier. The software belonging to the FPGA was first of all made for commercial FPGAs with a few modifications for this particular device. This meant that it was hard to know which part that was applicable for use in this case. For instance, could not System Designer itself be used but its included parts had to be used by itself instead. It is understandable that there is no software specific to such a small area but a better documentation could be suggested.

Something that also made the working load heavier was the compatibility issues found between the synthesis tool, Leonardo, and the place and route tool, IDS Figaro. When post-synthesis simulation was to be done with a design that included generated macros, some conversion had to be made by hand. Because the two tools use different ways of writing port names an underscore has to be added or removed, this can of course be done by a user created macro but it still involves extra work. No such macro was created during this project. There were also other observations that indicated misunderstandings between the software even though the real cause is not confirmed. What was blocking the process when this part of the project had to be abandoned was the misinterpretation of configuration signals in the SpW node design. In the SpW design pre-synthesis configuration of the node itself were done by signals and not generics, these signals were misinterpreted and could not be placed and routed. If the mistake is made in the place and route tool or already in the synthesis tool is hard to tell without a closer investigation.

Leonardo Spectrum were also providing some other inconveniences, the first one has no impact on the end result but the second one can be of more importance under more high demanding circumstances. There is a hitch with the “working directory” function, as the project was updated and eventually moved to a new folder the old working directory was still active locally even when the working directory were updated globally. This meant that by mistake the old design could easily be synthesised instead of the new design, which sometimes could be a difficult error to discover. The second problem is that the design is synthesised with a supply voltage of 4.75 V instead of 3.3 V which AT40KEL is using. According to the support from ATMEL this issue are taken care of during place and route, which is done at 3.3 V, but under high demand synthesis it can be beneficial to be aware of this problem.

The main objective was not to test the design of the SpW node, but one issue with the configuration is worth mentioning. As noted earlier in the report the setting of SYS_DEFAULT have some problems and does not pass the required tests. This setting is an unusual configuration of the clock domains and is not originally in the test, even if it is not specified in the documentation. If it is added to the test, errors are generated and the design cannot pass the test. This was not further investigated and the problem was instead solved by changing setting.

Another positive result was that a guide for a simple design flow procedure was developed, which contains several advice and guidelines on how to handle the basic processes. This guide can be found in appendix II in this report.

7 PERFORMANCE TEST

7.1 Additional Preparations

Since some of the preparations where the same for the SpaceWire test as for the performance test time could be saved. For instance a major part of the reading was already done and most of the lab equipment was already in place. One of the extra equipment needed was a signal generator which could go up to 60MHz. In the end the signal generator could not provide the frequencies promised so as an emergency solution a pattern generator was used as clock instead.

7.2 Execution

First of all a good test environment had to be created. In an early stage it was decided that to be able to load the Device Under Test (DUT) with data a test interface would be created internally inside the FPGA. This would include a UART for communication with a computer, a controller to write the data into the RAMs, another controller for carrying out the tests and obviously RAMs to store the data.

7.2.1 FUNCTIONAL DESCRIPTION OF THE TEST INTERFACE

A functional description of the Test Interface is presented here but there is also a User Guide produced for the Test Interface, presented as appendix I in this report. For more information on how to practically use the Test Interface see the User guide.

To interconnect all the different parts in the Test Interface a wrapper (UART_wrapper) was created. This was a pure wrapper with at most a couple of “and”- and “or”- gates to connect different signals to the same output in a correct way. The parts connected in the wrapper was:

- **miniUART**, a simple UART from www.opencores.org
- **UART_control**, a control that reads commands and writes data to the RAMs
- **run_test_ctrl**, control memory checks, run tests and read back the results
- **RAMs**, there are separate RAMs for storing data to test and to collect the results

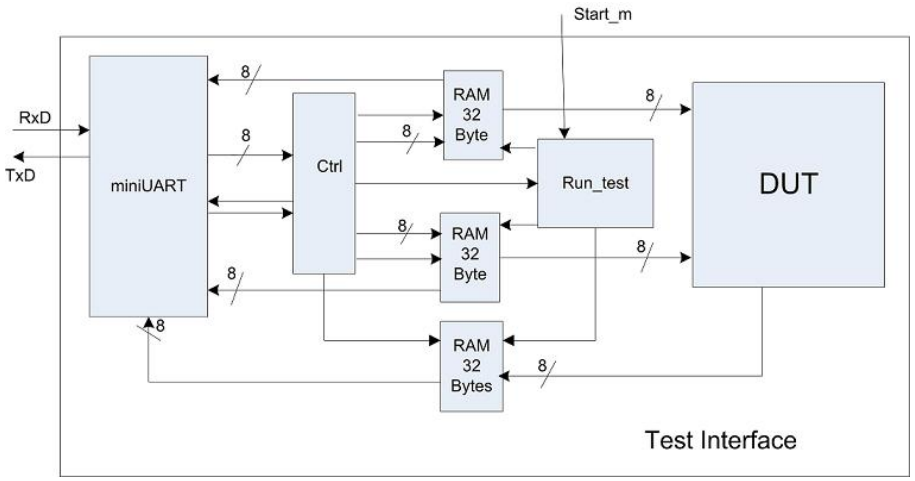

Figure 7-1. Test Interface

7.2.1.1 UART

The UART was not constructed from scratch but instead was acquired from www.opencores.org to save time. Even though this was a complete functional UART it had to be adapted to fit its purpose better which also made it easier to create a control for it. Some of the more noticeable modifications done were changing the reset to active high and removing the tri-states which would have become internal otherwise. The system clock was also changed from 40 MHz to 4MHz.

At the time the UART design was acquired from www.opencores.org no documentation was available so no references can be made. Instead a short overview will be presented here, based on experience. The UART is built up by four parts and a small package. One of the four parts was mainly a wrapper but since it also contains a control process it could be seen as a building block.

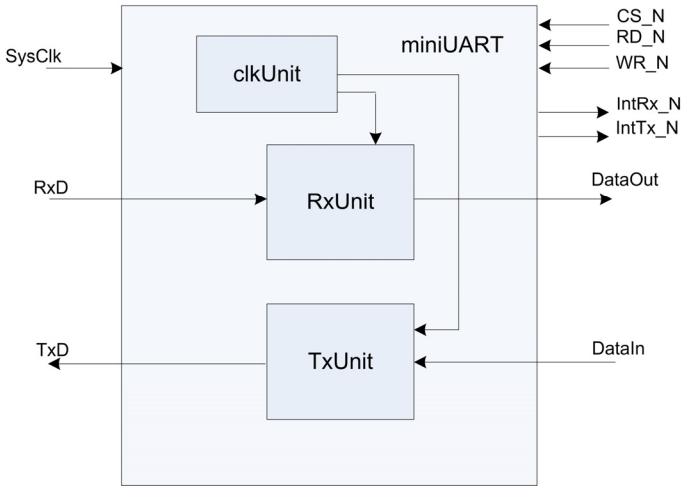

Figure 7-2. miniUART

7.2.1.1.1 clkUnit

The clkUnit receives the system clock (SysClk) and divide it to appropriate speeds for the enables of RxUnit and TxUnit. The design was originally made for a clock speed of 40 MHz and was divided down to 155 kHz for the receiver and to 9.6 kHz for the transceiver. This was modified to work at 4 MHz, which was instead divided down to the mentioned speeds. There was also reset functionality for resetting the internal counters. The reset was changed to active high.

Figure 7-3. clkUnit

7.2.1.1.2 RxUnit

RxUnit is designed to receive signals at 9.6 kHz. It uses the 155 kHz enable signal from clkUnit and samples 16 times and retains the value in the middle. The serial RxD signal is received in one end and after one byte has arrived it is put on the parallel output, DataIn. RxUnit also contains different status signals like Frame Error, Output Error and Received Data Ready (DRdy). These signals are used in a status register among other signals. DRdy is used in the control of the dataflow but the others are error flags and only showed on the output, never used in the control. RxUnit also has a "read" input which enables the receiver.

Figure 7-4. RxUnit

7.2.1.1.3 TxUnit

TxUnit receives the 9.6 kHz enable signal from SysClk and sends one bit every enable pulse as long as there is data to send. Data is loaded into TxUnit in parallel one byte at a time every time the load signal gets asserted. There is only a buffer of one byte, which means that one byte can be loaded while it is still sending the previous byte but if a second byte is loaded before the sending is done the first byte will be overwritten without a warning. TxUnit have also flags for when the buffer is empty and the output register is empty. There is also a synchronous reset in the design.

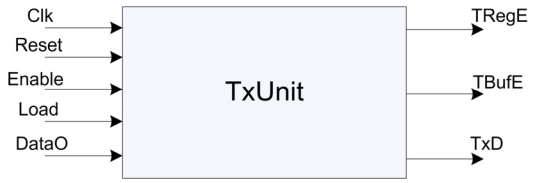

Figure 7-5. TxUnit

7.2.1.1.4 miniUART

miniUART is mainly a wrapper for the other parts but it also contains some control functions. In addition to connecting internal signals it combines the status signal into one status register, asserting control flags (IntRx_N, IntTx_N) and directs the external enables (CS_N, RD_N, WR_N). CS_N is the chip enable, RD_N enables the receiver and WR_N activates the transceiver. So if CS_N and RD_N are both enabled then the "read" signal to the receiver gets asserted. If CS_N and WR_N are active then the "load" signal gets asserted. IntRx_N is an interrupt control flag which is asserted when DRdy from the RxUnit is asserted which in principle means that it gives a pulse every time a byte is received. IntTx_N is an interrupt control flag and is asserted when the transceiver buffer is empty which makes it a good indicator when it is time to load the next byte. There is also some reset functionality in this design.

7.2.1.2 UART_control

UART_control consist of mainly two processes, one receives the bytes from the UART and keeps track if it is a configuration byte or data byte and one process that takes care of the incoming configuration bytes and sets the device in the right setting. The data is obtained from the signal "DATA_UART_IN when "INTRX_N" is active, both signals is received from miniUART. It has also two additional control inputs, one is a manual start taken from an input pin and a stop signal from run_test_ctrl. UART_control also receives the system clock (SYSCLK) for clocking and RESET resets the system. Since this is the control segment of the design many control flags are going out from this part. There are some addresses and write enables to RAMs, a MEM_CHECK flag for memory data check, READ_RESULT flag for reading back the results through the UART and START flag for starting a test via command. There is also two control signal sent back to miniUART that is always kept active and it is CS_N and RD_N. More information on how these signals are handled/produced is given in the closer description of their various processes.

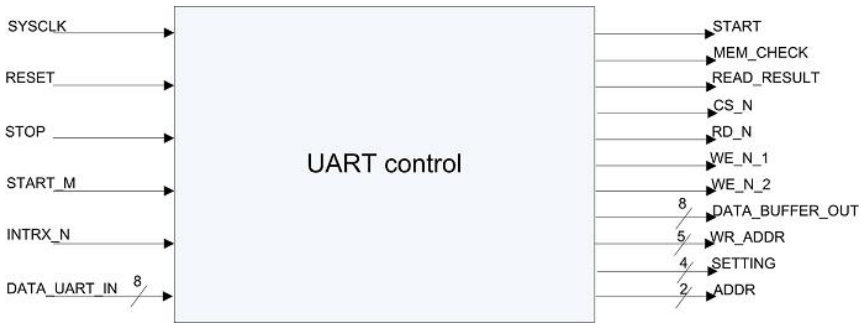

Figure 7-6. UART_control

7.2.1.2.1 The "data_in" process

First of all this process has an asynchronous reset, which resets various internal signals addresses and control flags. If no reset signal is active it is a synchronous process that is waiting for INTRX_N to be asserted, when that happens it means that it is time to read a byte. If this is the first byte coming in then it is a configuration byte. When a configuration byte is received it sends out a flag (CON_F) to the controller process which read in the byte and configure the setting accordingly. In the configuration byte information can be given on how many of the following bytes is considered data bytes and should be written to memory instead. If that is the case then when the next byte comes in, a counter will start counting the bytes received and instead of sending the data to the controller it is written to the two RAMs for incoming data. When all the expected data bytes are acquired the process expect the next byte to be a configuration byte, and so it continues. There are also two options on how to write the data to the memories, the default setting is that the first half of the data is written to RAM_1 and then the other half to RAM_2, but there is also the possibility to write the same data to both RAMs. It is then of course important not to send to much data to the memories since only half the amount of data is needed.

7.2.1.2.2 The "controller" process

This process has also an asynchronous reset but is otherwise synchronous. It awaits the CON_F control flag from the "data_in" process, which signals that it is time to read in a configuration byte. If the data length, DL, part of the byte is "0000" then it is in configuration mode and indicates that the control section, CS, is giving a command instead of deciding which test to run. Since no data is expected after this command, the next byte received must also be a configuration byte.

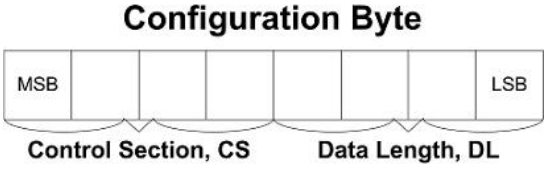

Figure 7-7. Configuration Byte

Otherwise the DL section is usually for deciding how many data byte there will follow and CS section which test to run. In the current design the CS section is not that important when data is loaded and is only important when the run test command is sent. The CS section is however sent to the run_test_ctrl part to inform which test to run.

7.2.1.3 run_test_ctrl

This part consists of four synchronous processes and a small amount of parallel logic. It also receives many of the control signals from UART_control, like MEM_CHECK, READ_RESULT, START and SETTING. These signals decides if the input data should be sent back and checked, if the results should be read back, which test to run or if the test should be started. Run_test_ctrl also acquires a signal from the miniUART, INTTX_N, indicating if the UART is sending for the moment or if a new byte can be sent. The standard signals SYSCLK and RESET are also received. For outputs there are address and enables both for the RAMs with the input data and for the one collecting all the data. In the special case when the DUT is a RAM the control for that RAM is also run from this part. There is also a STOP signal that is asserted when the test is done and a MEM_SEL signal that which specify which RAM to read from during the memory check. This part is also responsible for asserting WR_N for miniUART when something is sent back through the UART.

Figure 7-8. run_test_ctrl

7.2.1.3.1 The "run_test" process

The process begins with an asynchronous reset and is synchronous in all its remaining functionality. The synchronous part starts with checking if the "in data" in the RAMs should be controlled for errors. If MEM_CHECK indicate that this is so, then the first byte is sent to the miniUART and a counter starts to count the bytes sent. The process is coded so that it will not send another byte to miniUART until it has stopped sending to the computer. This is implemented by an internal ARM signal that after sending a byte will not "arm" again until INTTX_N has stopped being active. As a result of this the buffer is not fully used, but almost no performance is lost because the system clock is much faster than the transmission speed. When all 64 positions are sent the design stop the "test" and awaits the next command. An important note is that all data in the

RAMs are read independent on how many bytes of relevant data there is. The memories are also read one after another, in the same way as it is written so if the RAMs are not filled then there will be bytes of irrelevant data between the wanted data bytes.

If MEM_CHECK is not asserted then the process looks to the SETTING received from UART_control and runs the specified test. At the present time there are only two different tests that can be run. The default one is 32 runs with 8 bits that means that 32 bytes each are sent to the two separate outputs, which in turn is fed to the DUT. The result is then collected in another RAM through a single bus. This setup is very suitable for test on DUTs like adders and multipliers, but there is also another version that is made for RAMs as DUT. That version uses the same address and enable signals that are used for reading from the "data in" memory and in that way creating a control for the DUT RAM. In the case of a DUT RAM only the first "data in "RAM is used. The second test setting that can be run is when only the first position in both RAMs is used and in all the other aspects it is the same as the default setting. This results in merely one run instead of 32. Both these test are sending a STOP signal when the test is done so that the design is open for new tasks. When a test has started, only a reset can stop it before it is done, this should however never be a problem since the test finish in micro seconds. A good benefit of this feature is that it is possible to use a non spike free input for the manual start without getting unreliable results, the last "spike" will always produce a clean test.

7.2.1.3.2 The "read_DUT_data" process

This is a synchronous process that writes the results from the DUT into a RAM. There are two versions of this, one default version where the result is read under the same clock cycle as it is fed into the DUT and a special version for when the DUT is a RAM. Both versions include an asynchronous reset.

In the default version it operates in a very straight forward way, by using the same addresses and enables for writing into the result RAM as is used for reading from the "data in" RAM.

In the special version it manages this in another way. Instead of receiving the results at the same time as it is written into the DUT RAM, it waits until the test has finished and then transfers the test data to the result RAM. This is also done in system clock speed, which makes it a part of the test. In this way both read and write is tested.

7.2.1.3.3 The "send_result" process

Like most other processes in this design, this is a synchronous process with an asynchronous reset. The process awaits the "READ_RESULT" signal to go active, when it is asserted the data in the result RAM is sent back through the UART to the computer. This is done with the same principle as when the data in the "data_in" memories are checked through the UART, but since it is only one RAM instead of two there is a little bit less logic.

7.2.1.3.4 The "run_ctrl" process

Run_ctrl is a very small process and does not even have a reset, since it is not needed. The only thing this process control is the RUN signal. It receives the START signal and when it gets asserted the process assert RUN and when the STOP signal becomes active, the RUN signal is unasserted.

7.2.1.4 RAMs

There are several RAM in the design and the number can vary depending on which kind of DUT to test. At the present time the different designs for the test are not integrated into a larger design, instead there is a special device for every test. This is most evident when it comes to RAM since it is a feature that is changed the most, depending on the test is 8 bits, 16 bits or if the DUT produce results that require a larger result collecting RAM. In the basic design there are two memories at the input receiving the data for running the test ("data_in") and one RAM to acquire the data from the DUT during or after test.

7.2.2 TEST PROCEDURE

Much of the previous set up could be used in this case. The computer was already connected to the board through a parallel cable and the software to download the design was already installed. The logic analyser and pattern generator was also available but needed to be connected to the right pins on the board. In addition to a small circuit to convert the UART signals to 3.3 V, there was also a need to install software in the computer to communicate through the UART with the test interface.

The UART converter circuit was connected through an ordinary serial cable to COM2 port on the computer in one end. In the other end the circuit was connected to the board with the pins for the receiver and transceiver as well as the ground and supply. The connectors were not there on the ordered product and had to be soldered there on its arrival.

Signals were only sent by the Pattern Generator and received by a Logic Analyser during test without the Test Interface. When the Test Interface was in use all communication was done through the UART. The Pattern Generator was connected to the input of the DUT but since the different DUTs did not have the exact same pin configuration, small changes had to be done between tests. These changes were unavoidable but they were kept to a minimum to reduce extra work. The same problem applies for the output pins that are connected to the Logic Analyser. A feature in the software were helpful here, it made it possible to import an incomplete pin connection scheme which could then easily be updated. It is possible to read more about this in the appendix II, Design Flow Guide.

The software used to communicate through the UART was Terminal v1.9b, a freeware. The commands received by the Test Interface were in binary code but the information sent by Terminal was described in hexadecimal so a simple conversion has to be made. Terminal could also send files of data, so the practice used was to prepare files with commands and data to import. These files were written in the software "Free Hex Editor" which easily could prepare hexadecimal data files. The data received by Terminal could be in binary but as the familiarity with the "hex-commands" was better than the binary commands, "hex" was preferred.

When everything was set the first thing done was to reset and clear the design. The reset does not clear the memories so when the design was reset, zeros were loaded into the memories and driven through the design so that all RAMs had zeros at all positions. When all the RAMs were cleared a test sequence of data where sent to the first pair of memories. This was usually 1 to 32 or 64 in ordinary binary form or in Grey-code. So far everything has been done in 4 MHz, the speed the UART interface was designed to work at. Now when the data is ready to go through the DUT the clock speed were changed to the speed that was going to be tested. With a new system clock speed the test was started manually by asserting a pin. The results were now collected and saved in a RAM waiting to be retrieved. The system clock speed was once again set to 4 MHz and the result where received through the UART and displayed on the computer screen. The RAMs with the result were read two times after each other to confirm that there were no bitflips when the data was transmitted. During this project there were never any bitflips found during UART communication. As a temporary solution the Pattern Generator provided the system clock, since the Signal Generator could not provide the speeds it was specified for.

When the Test Interface was not used, signals were instead treated with the Pattern Generator and Logic Analyser. In this case the procedure was simple, first of all a pattern of signals were created in the Pattern Generator and sent through the DUT and then collected at the output pins by the Logic Analyser. The result could then be analysed usually by manual observation on the screen of the Logic Analyser. Many designs were prepared according to the project plan but because of unfortunate events and lack of time not many results were produced, see the results section for more information.

According to the project plan many designs were to be tested but as mentioned above not many designs were tested in the end. Mainly two designs were tested and some additional designs were prepared. The unregistered adder was to be tested in both test approaches. An adder with registered outputs was also designed to be tested in the same way as the unregistered one. The settings chosen when these DUTs where generated can be seen below.

Unregistered Adder

Carry In: Disabled
Carry Out: Disabled
Register: None
Signed Overflow Pin: No
Width: 8
Pitch: 1
Aspect Ratios: 0.00
Macro Name: add8_no_reg

This gave following size and speed described by the generator:

Logic Size: 1 x 8 logic cells
Max Speed: 101.6 MHz

Adder with Registered Outputs

Carry In: Disabled
Carry Out: Disabled
Register: Output
Width: 8
Pitch: 1
Aspect Ratios: 0.00
Initialization: Reset
Macro Name: add8_regout
Initialization Polarity = Low: No

This gave following size and speed described by the generator:

Logic Size: 1 x 8 logic cells
Max Speed: 118.3 MHz

The RAM design was mainly designed to be tested with the Test Interface but its outputs were also tested manually in a test resembling the test above. For the test without the Test Interface additional designs were also prepared and that was signed and unsigned multipliers as well as a pipelined multiplier. Unfortunately these were never tested.

7.3 Results

As described in “Test Procedure” two different kinds of test were made on the device. The results from these two tests are described under its own headline.

7.3.1 TEST WITH TEST INTERFACE

The results are only acting as a guideline on what can be achieved. There was no time in this project to optimise the Test Interface so that the DUTs could be pushed to the limit during test. The tests have been done on small number of samples which also limits the precision of the test. With this in mind the test shows that a design can be loaded to the FPGA in a successful way, and the design can also communicate through a UART in a satisfactory level.

The first test preformed was to test the internal RAMs. Since the Test Interface included RAMs and they were deemed to be the bottleneck when it comes to system clock speed in the test feature (part) of the interface, it was important to test them first. The data sheet (AT40KEL040, 2006) states that the access time for the internal RAM is 18 ns, (two different values for the maximum speed of the RAM is stated in an older version) which will limit the memory to a clock speed of around 55.6 MHz. The overall maximum internal clock speed achievable for the FPGA is 60 MHz according to the data sheet, which puts the RAMs slightly lower. By having a RAM as DUT a maximum speed can be tested, this can then be used in the following tests. If the DUT is changed to something else and it breaks down below the limit achieved with the RAMs it can be assumed it is the DUT breaking down. But if the test instead breaks down at the limit or above it can be assumed that it is probably the Test Interface that gives up.

The RAM was tested a little bit differently than other tests. First of all, only one of the input memories were used and at the first clock pulse the data was read from the input RAM to the DUT RAM. On the second pulse the data from the DUT RAM were written to the result RAM. Instead of only excluding the DUT and read from the input RAM to the result RAM directly this “extra” RAM was included to add some additional read and write steps, which would make it easier for errors to manifest when the test was run on its limits.

As mention earlier the test does not include many samples since all of them have to be checked by hand. When no errors were detected on 5 x 32 bytes, the design was considered to work well enough. In the following diagram the number of faults at each frequency can be seen. This gives merely a simplistic view, since readings that contain many faults could actually be a result of one fault that created a shift through the whole test sequence. While other tests with fairly many faults could have several individual faults. The highest speed that did not result in any errors was 46 MHz, which is reasonable under these circumstances.

Test with RAM					
MHz	1st	2nd	3rd	4th	5th
4	0				
10	0				
40	0	0			
45	0	0			
46	0	0	0	0	0
47	1	1	0	0	1
48	25	9	0	1	26
50	8	7			

Table 7-1. Test with RAM

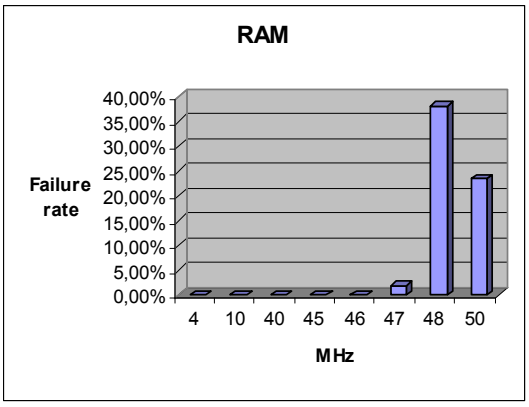

Figure 7-9. Test with RAM

It can be seen in the table that the results are irregular and it is far too few samples to define a certain failure rate. What the graph does is only to display the data in a more accessible way. The picture is not as easy at it seems in the results, for instance the first and fifth reading at 48 MHz has a very high failure rate because whole series have been shifted due to an initial error, while at other times errors can be spread out individual errors. It most be said though that the most common errors were detected in bursts.

The RAMs was then also tested with grey-coded data to avoid racing and see if there is a difference between the results. In the end it does not seem like racing play a major role since the limit now was at 47 MHz, which is a little bit higher, but the small variation can as well be due to some other cause. With such a few samples it is very hard to say if there is a difference.

Test with RAM, Grey-code					
MHz	1st	2nd	3rd	4th	5th
4	0				
40	0	0			
47	0	0	0	0	0
48	0	0	9	0	0
49	7				
50	7				

Table 7-2. Test with RAM, Grey-code

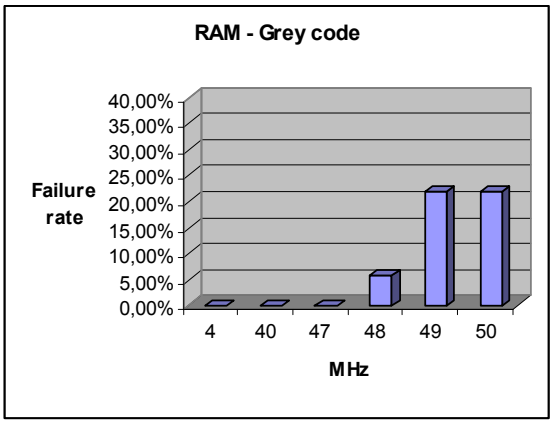

Figure 7-10. Test with RAM, Grey-code

After the initial test of the RAMs it was time to change the DUT. Instead of having a RAM as DUT an unregistered adder was tested. Now data was taken from the two input RAMs and added so the result could be written into the result RAM under the same clock pulse. The result from this test was first surprising since better results were expected but after examining the procedure the results are a bit more understandable.

The design was now functional up to only 32 MHz which is far lower than for the RAMs. This can probably be explained by that the signals had to travel further per clock pulse than with the RAMs, even though the RAM test included more elements. In the "adder test" the signals had to go from the input RAM, through the adder, and then be written to the result RAM in one clock pulse. While in the RAM test the signal had only to go from one RAM to another in one clock pulse, even though it included more transitions. But this may not be the only issue, since the two test designs are synthesised and placed and routed separately it is hard to tell if the designs are exactly the same. Because of the problems with putting proper constraints during development procedure, this problem could not be verified within this projects timeframe. These are the results:

Test with Unreg Adder

MHz	1st	2nd	3rd	4th	5th
4	0				
10	0				
30	0				
32	0	0	0	0	0
33	4	4	4	4	3
35	4	4	5		
40	14	14			
45	32				

Table 7-3. Test with Unreg. Adder

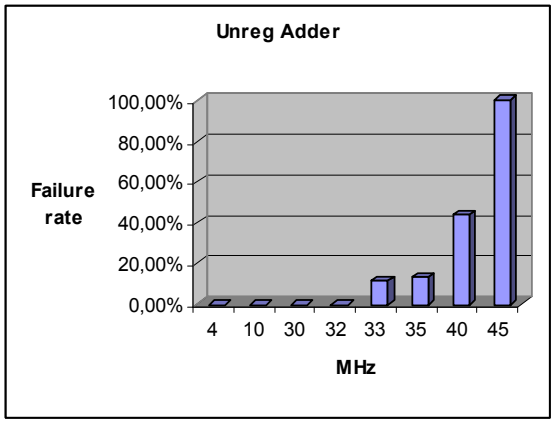

Figure 7-11. Test with Unreg. Adder

To test if the result could be improved by more demanding constraints, a small test was done with a very tough constraint on the system clock. In the end this constraint were not met but could push the tool for higher performance. But the result did not show any signs of improvement, which indicates that the problem is not that easy. The results are shown below:

Test with Unreg Adder, Higher Constraints

MHz	1st	2nd	3rd	4th	5th
4	0				
31	0	0	0		
32	1	1	1		
33	2				

Table 7-4. Unreg. Adder, Higher Constr.

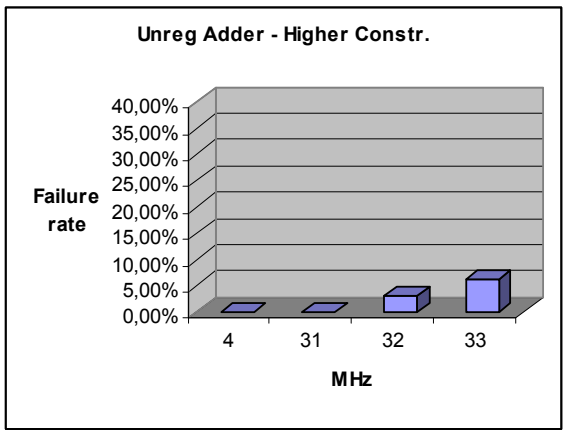

Figure 7-12. Unreg. Adder, Higher Constr.

These results make it very interesting to see how an adder with registered output would perform, since it shortens the length the signals have to travel. If it performs better than the unregistered adder it is possible to get a feeling on how large impact signal travel length has on the performance in this particular case. Unfortunately, this test run into design issues in the very last stage of the project and had to be abandoned due to the lack of time.

In addition to these results mentioned above there were also some additional issues found in the software. There are essentially two problems and the first one has to do with the enables for the internal RAM. When a RAM that was generated in IDS was inserted into the design the enables got inverted, which obviously made the design non-functional. The second issue was with the software's interpretation of the port map, in some cases Leonardo's schematics show ports as if they were not connected. The post synthesis simulation still works but when the design is loaded into IDS Figaro problems occur and the tool complains about ports not existing. The reasons for these two problems were never found but they could be worked around.

7.3.2 TEST WITHOUT TEST INTERFACE

Testing without the Test Interface brings many benefits since it does not have to adhere to the limitations of the Test Interface itself. This would in the end give the most reliable results if only the problem of visual inspection could be solved. Since the results would be evaluated manually on the screen of the Logic Analyser it is hard to define the transition from good to bad signals. Otherwise it is few aspects to consider when analysing the results compared to the Test Interface, which gives a much more complicated picture. A more straight forward approach is traded with the lack of automation and long run time which could be provided by a test interface.

Unfortunately the timeframe did not allow much testing and only a few results were produced. In figure 7-13 test signals from a RAM test at 55 MHz can be seen, this was also the only test made with satisfying results. The reason for this was that the testing had initially some problems and before some of them could be solved the FPGA stopped working. The RAM test showed that the output gradually worsened between 50 and 55 MHz and at 55 MHz the functionality were lost for certain. There was also a test performed on an adder with registered outputs, but the test never presented any reliable result. The test phase had then to be abandoned regrettably, due to the lack of a working FPGA and the short timeframe.

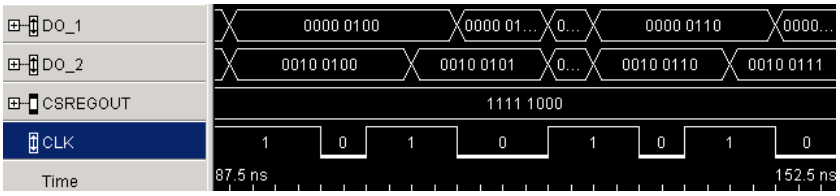

Figure 7-13. Test Signals on a RAM at 55 MHz

8 CONCLUSIONS

The results from this project are by themselves not a satisfactory description of the FPGA and its software. But the test of the software that the first part of the project provided in addition to the functioning design in the FPGA that the second part demonstrated, it can be said that the main objective have been fulfilled. Together these two parts have been able to give useful feedback about the implementation process at the basic level.

The overall conclusions from this project are divided into three distinct areas, which are the Evaluation board, the FPGA and the Software. Apart from these areas it is also important to mention the support from ATMEL, since their quick and helpful answers have been of great importance at several occasions.

The software was the first area encountered in this project and as can be read in the results it was fairly hard to get a good understanding of the software initially. This was mainly due to the problem to know what features were applicable for the Rad-Hard version of the FPGA. The software also showed signs of other problems, like compatibility issues, but most of them could be solved or avoided thanks to the help from the support.

The evaluation board was of a simple design with almost only the FPGA and its configuration and power circuitry mounted. This gave a very easy access to all the pins of the FPGA, which made probing and testing easier. Even if this simple approach made the probing easier a development board with extra features would ease the integration of an application, like the optional last step of integrating the SpaceWire in a SpaceWire network. These features could be external RAMs, spike free switches or LVDS outputs.

Also the FPGA had very clean characteristics, without many extra features. This can put some limitations on some designs but the FPGA included rad-hard internal RAMs, which is an important attribute. Something that would improve the RAM even further would be two separate clock inputs for the dual port RAMs. With only one clock input the RAMs cannot function as a clock domain interface as easily as it would have done otherwise. The problem can be worked around but it is convenient to use the RAMs as interfaces. It is also worth mentioning that the FPGA always booted correctly. There were some cases where errors occurred while loading the design from the computer, but these cases were always detected and it was working after a retry.

9 DISCUSSION AND FOLLOW UP

Since there is few users of this product at this stage the feedback from this project has been useful in ATMELs development of future products in addition to its use within ESA. Already today it is known that the next product from ATMEL, AT280F, will include features asked for in AT40KEL-DK. There will for example be a better equipped board to mount the FPGA in with external memories among other things. The software will also be updated and the new synthesis tool from Mentor Graphics, Precision, will replace Leonardo Spectrum as well as there will be an updated version of IDS Figaro. The next generation will also be more than four times as big which will allow for more complex designs. Most of the internal structure is the same however, so there will still not be two clock inputs to the dual port RAMs.

One important factor is also that this FPGA is based on European technology which means it will not be subject to the ITAR restrictions that the U.S. government has put on U.S. exports. This is good for the market outside USA as it will be easier to buy these products. It is also good for the European market to be able to produce such technology.

The project had to be stopped in its most interesting phase before most of the results could be produced. That means that many results can be produced with relatively little effort from where the projected ended. Because of this there is a wish and hope that this work will go on and continue where this project ended.

10 REFERENCES

Books and papers:

Balch, M., (2003). *Complete Digital Design (ebook)*, McGraw-Hill, pp. 222, ISBN: 0-07-143347-3

Bonacini, S., et al. (2006). An SEU-Robust Configurable Logic Block for the Implementation of a Radiation-Tolerant FPGA. *IEEE Transactions on Nuclear Science*, Vol. 53, No. 6, pp. 3408-3416.

Pedroni, V. A., (2004). *Circuit design with VHDL (ebook)*, Massachusetts Institute of Technology, pp 3, ISBN 0-262-16224-5

Pellerin, D. & Thibault, S., (2005). *Practical FPGA Programming in C (ebook)*, Prentice Hall PTR, ISBN: 0-13-154318-0

Tiwari, A. & Tomko, K. A., (2005). Enhanced Reliability of Finite-State Machines in FPGA Through Efficient Fault Detection and Correction. *IEEE Transactions on Reliability*, Vol. 54, No. 3, pp. 459-467.

Sterpone, L. & Violante, M., (2005). Analysis of the Robustness of the TMR Architecture in SRAM-based FPGAs. *IEEE Transactions on Nuclear Science,* Vol. 52, No. 5, pp. 1545-1549.

Online material and documentation:

AT40KEL-DK Design Kit User Guide, (2006). Atmel Corporation. Available: www.atmel.com.

AT40KEL040 (datasheet), (2006). Atmel Corporation. Available: www.atmel.com.

ECSS-E-50 -12A, (2003). European Cooperation for Space Standardization (ECSS). Available: www.ecss.nl for a free registration.

Microelectronic Section's Webpage, *Synthesizable IP-Cores Available from ESA (Online)*. Available: http://microelectronics.esa.int/core/corepage.html, 2007 February 10th.

SpaceWire Codec VHDL User Manual, (2005). University of Dundee. Available: http://microelectronics.esa.int/core/ipdoc/uodspacewire/SpaceWire-Codec-VHDL-User-Manual.pdf, 2007 February 11th.

SpaceWire Link Interface RTL Verification, (2005). University of Dundee. Available: http://microelectronics.esa.int/core/ipdoc/uodspacewire/SpaceWire-Link-Interface-RTL-Verification.pdf, 2007 February 11th.

esa DOCUMENT

document title/ titre du document

User Guide for the Test Interface

APPENDIX I

prepared by/ *préparé par* Hakan Helzenius

reference/ *réference*
issue/ *édition* 1
revision/ *révision* 0
date of issue/ *date d'édition*
status/ *état* Draft
Document type/ *type de document* Technical Note
Distribution/ *distribution*

European Space Agency
Agence spatiale européenne

ESTEC
Keplerlaan 1 - 2201 AZ Noordwijk - The Netherlands
Tel. (31) 71 5656565 - Fax (31) 71 5656040

Table Of Contents

List of Figures

List of Tables

1 INTRODUCTION

The Test Interface is a small design developed during a project to test the performance of a development kit including an SRAM based FPGA from ATMEL, AT40KEL. Parts of the testing of the FPGA were done by an internal Test Interface inside the FPGA, which communicated with a computer through a UART. The internal structure of the Test Interface is better described in the project report, Performance Test of AT40KEL-DK, while this document is dedicated to a user description. This Test Interface is a very small design with only one purpose but with a possibility to be upgraded and extended.

2 OVERVIEW

The Test Interface receives data and commands from a computer through a UART. When the Test Interface is communicating with the computer it is working at 4 MHz but is sending at 9.6 kHz over the serial connection. When everything is prepared for a test the UART is not needed anymore and the system speed can be changed. At other clock speeds than 4 MHz the few commands necessary are operated manually instead of from the computer. After the test when it is time to retrieve results, the system clock is set back to 4 MHz and the results are sent to the computer. This procedure is done to be able to stress the clock speed over the DUT to its limits while the UART only works at 4 MHz. As can be understood from above the Test Interface can be divided into two main parts, the interface towards the computer and the external environment and the part for testing. The overall design is described in the picture below.

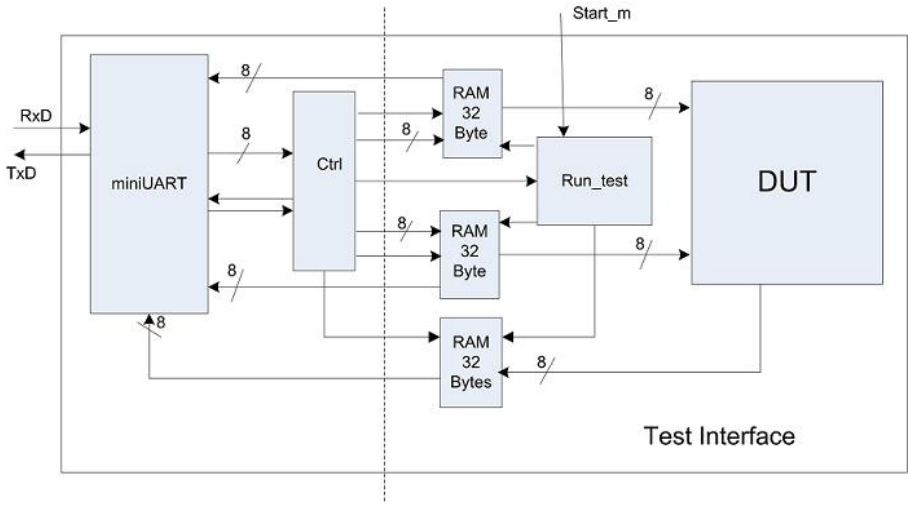

Figure 2-1. Test Interface

3 PHYSICAL LAYER

The design itself does not really contain a physical layer except the FPGA it is loaded into, but there is a need for some additional equipment to be able to perform a test. Apart from the FPGA and the evaluation board it is mounted on, a computer, a serial cable and a UART converter is needed. The UART converter adapts the board's 3.3V CMOS signals to the standard serial connector signals. See below for a figure of the configuration and a table summarising the equipment.

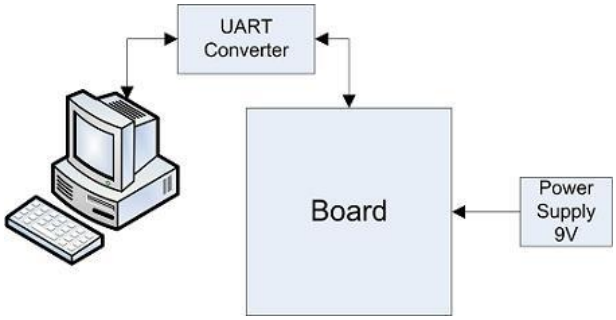

Figure 3-1. Hardware Configuration

Equipment used	**Description**
Mother- & Daughterboard for the FPGA	From ATMEL
Power supply for the motherboard	9 V
UART converter	
Standard serial cable	Male - Female
Computer	PC

Table 3-1. Equipment

4 SOFTWARE AND START UP

To be able to communicate with the Test Interface the computer need to have access to certain functionalities. It needs to have "bit-control" over the signals sent through the serial port. This can probably be done by the Hyperterminal in Windows, but this project used a freeware, called Terminal v1.9b, because of its better access to single commands. In Terminal there was the possibility to write short sequences in hexadecimal code in the software and then simply push send. But there was also the option to prepare sequences in files and import them, which is very helpful when data has to be sent. The files that were going to be imported to Terminal had to be created in another software able to create .hex files, for this a different freeware, Free Hex Editor, was used. In Free Hex Editor longer series of data could easily be written and saved in .hex format among other format options. Below a figure of the Terminal tool is shown as well as a list over the software. In the figure it is also possible to see the settings used during test.

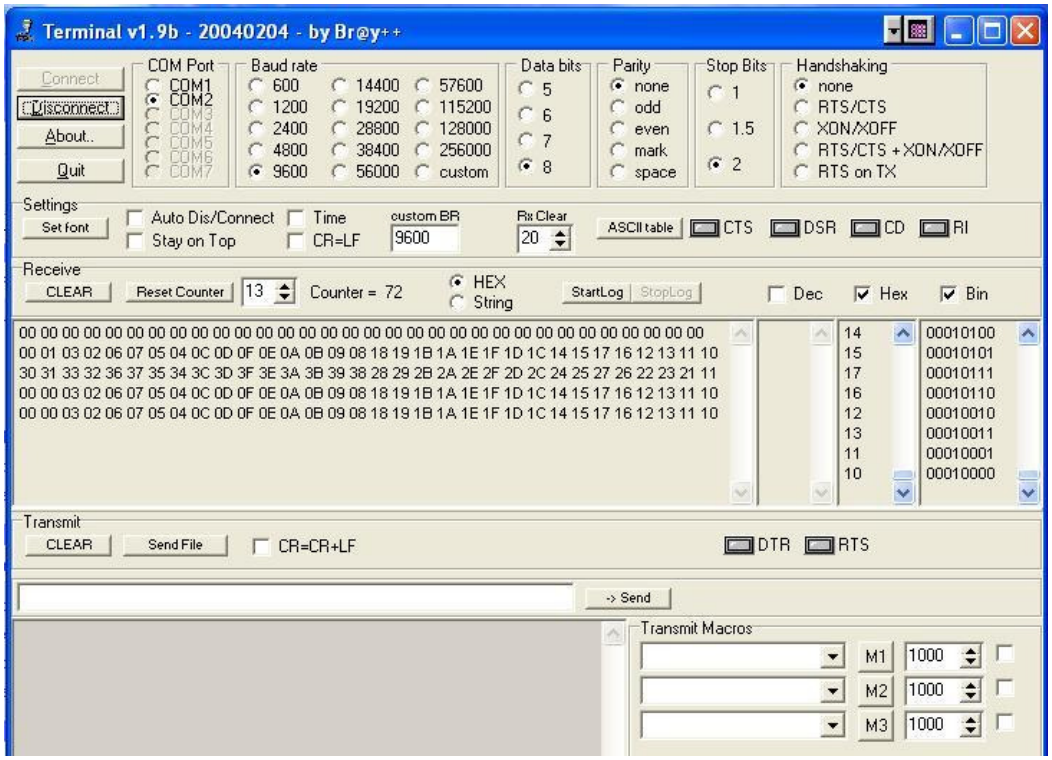

Figure 4-1. Terminal v1.9b

Software used	Description
Terminal v1.9b	To send and receive signals over the COM-port
Free Hex Editor	To prepare files to be sent

Table 4-1. Test Software

The baud rate could be changed to a higher speed, even if it meant minor changes in the Test Interface design, it is not recommended. Since there is no handshake in use a higher baud rate increases the risk of transmission errors, which would be an unnecessary vulnerability when sending test results. There are not any great benefits of a higher transmission rate because of the relatively low amount of data that needs to be sent.

As can be seen in the picture above two stop bits are used instead of the usual one. For short commands it is possible to use only one stop bit but when series longer than a couple of bytes has to be sent, two stop bits are recommended to avoid errors. Since this means only an insignificant difference in functionality there were not any focus on solving this issue.

During the tests several files with commands and data series were created. These files are kept and can be provide for additional testing. The files available right now can be seen in the table below along with a short description.

HEX Test Files

Name	Description
RB	Read back input RAMs
RUN	Run default test
RR	Read back results
2B	Send 2 data bytes
2B_RB	2B followed by RB
6B	Send 6 data bytes
64B	Send 64 data bytes (Grey code)
64B_RB	64B followed by RB
64B_RB_noGrey	64B (no grey code) followed by RB
64B_RUN_RB	64B followed by RUN and then RB
64B_RUN_RR	64B followed by RUN and then RR
NULL_64B	Send 64 nulls to reset the RAMs
NULL_64B_RB	NULL_64B followed by RB
NULL_64B_RUN_RR	NULL_64B followed by RUN and then RR

Table 4-2. Test Files

5 COMMUNICATION PROTOCOL

The protocol used with the Test Interface was created as an ad-hoc solution for this purpose. Originally it was created for many more features but in the end only a few were realised. The main idea behind the protocol was to have something simple to transfer data and commands. The approach chosen was to start with a configuration byte which stated how much data was going to be sent and when that amount of data was received the Test Interface was once again waiting for a configuration byte. This way of working mainly put the work load on the FPGA decoder side, while having a human interface on the other. The first byte was divided into two major parts, see figure below.

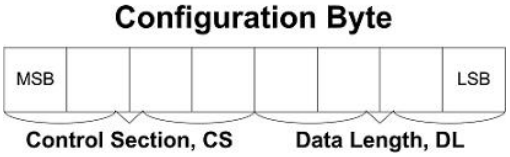

Figure 5-1. Configuration Byte

The “Data Length” part of the configuration byte defines how many data bytes there are to expect after this byte. The other section of the byte is called the “Control Section” and it decides which test to run. But if the DL part is set to zero, “0000”, the design goes into Config Mode instead and then CS is interpreted as a different set of commands. The chart below shows which functions were planned from the beginning and which were realised in the end.

Data Length, DL

Command	Description	Functional	Comments
0000	Enter Config Mode	Yes	CS from same byte counts as in Config Mode
0001	1 byte	Yes	Not recommended without "Input Mem Config"
0010	2 bytes	Yes	
0011	4 bytes	Yes	
0100	6 bytes	Yes	
0101	8 bytes	Yes	
0110	10 bytes	Yes	
0111	16 bytes	Yes	
1000	24 bytes	Yes	
1001	32 bytes	Yes	
1010	64 bytes	Yes	
1111	Start Test	Yes	Works only at 4 MHz test speed

Table 5-1. Data Length commands

Control Section, CS, in normal mode

Command	Description	Functional	Comments
0000	1 time, 8 bits	Yes	Runs one test with 8 bits width
0001	32 times, 8 bits	Yes	Default, Runs 32 tests with 8 bits width
0010	1 time, 16 bits	No	Runs default instead
0011	1 time, 32 bits	No	Runs default instead
0100	32 times, 16 bits	No	Runs default instead
0101	Continuous, 8 bits	No	Runs default instead
0110	Continuous, 16 bits	No	Runs default instead

Table 5-2. Control Section commands in normal mode

Control Section, CS, in config mode

Command	Description	Functional	Comments
0000	Idle State	Yes	
0001	Check Input	Yes	Reads back from the input RAMs
0010	Input Mem Config	Yes	Writes the same data to both input RAMS
0011	Read Results	Yes	Reads back the results
0101	Stop	No	Stops a continuous test run
1000	Reset	No	Reset by command

Table 5-3. Control Section commands in Config mode

In most cases 64 bytes were sent to the test Interface, 32 bytes to each Input RAM, and the default test was run. Next chapter will give examples on how these commands can be used and walk through the test sequence step by step.

6 TEST SEQUENCE, STEP BY STEP

This chapter will give an example on how a normal test sequence could look like and walk through the procedure step by step. What will be described below should only be a guideline on what can be done, feel free to make improvements and work out other ways to use the Test Interface.

First of all everything has to be connected the right way. See to it that the UART converter is connected to the right pins on the board, there are only one pin for receiving, one pin for transmitting and one pin each for ground and supply. Then a standard serial cable is needed to connect the UART converter with the computer. If every device has power and the design is loaded into the FPGA correctly, there should not be any more hardware preparations. The pins used in this project can be seen in the table below or in the example later on.

Pin connections	
I/O303	Reset
I/O298_A3	RxD
I/O297_CS1_A2	START_M
I/O292	TxD
I/O288_GCK6	SYSCLK

Table 6-1. Pin connections

If appropriate software is available in the computer it is time to start testing the design. Test commands for reading back from input RAMs, run test and read back result. Try also to reset the memories by filling the RAMs with zeros and drive it through the test sequence. When all of this is working and the RAMs are reset a set of test data can be written to the input memories. If the test is going to be run at another speed than 4 MHz the system clock speed is changed to the appropriate frequency. After that “start” is activated manually by asserting a pin and after waiting a couple of seconds the clock speed is changed back to 4 MHz. The result can then be read back through the UART and analysed. An example below goes through the whole testing procedure step by step.

Example: Test of an Unregistered Adder

After the design is prepared and loaded into the FPGA it is time to set up the test.

The UART converter is checked so that it is connected correctly. In this case the receiver input of the Test Interface is at pin I/O298_A3 and its transceiver output is at pin I/O292. The converter is also connected to ground and power supply through pins on the board. In addition to that it is connected to computer by an ordinary serial cable.
The power supply of the board is turned on and if the computer is off, it is turned on as well.

The program Terminal v1.9b is started and if the already prepared "data"-files is not enough Free Hex Editor is also started. The prepared files can be seen in table 4-2.

The system clock was started at 4 MHz, in this case by a pattern generator.

The design was then reset by having the reset pin connected and unconnected, pin I/O303 in this design.

Some initial test communication is carried out to see that everything is working before the memories are reset. A command sequence could look like this RR, 64B_RB, NULL_64B_RB and NULL_64B_RUN_RR. If the RAMs do not have zeros on all positions after the last command NULL_64B_RUN_RR is run again.

To send data and commands via Terminal the hex-code were either written in the "Send window" which were followed by a click on the "–>Send" button, or more commonly a file were sent. To send a file the "Send File" was simply pushed, where the wanted file could be chosen.

If new files had to be created in Free Hex Editor the data or commands were simply written in the editor and saved as a hex-file. If the file were to be used directly after its creation it was made sure that Free Hex Editor were closed down since it otherwise produced a conflict with Terminal.

After the RAMs been reset 64B_RB were sent to load the input RAMs with data and read it back to control that nothing went wrong.

The system clock was then changed to the test frequency, for example 47 MHz.
This was followed by manually asserting the "Start_m" for a short while, to run the test. In this case pin I/O297_CS1_A2.

After the test run, the system clock was changed back to 4 MHz to allow for the results to be sent back.

To send back the results the file RR was sent and the results could then be read in the Terminal's "receive screen".

The results were studied manually and to avoid that errors during transmission would corrupt the data, the results were received two times from the result RAM and compared. There were never, under the whole project, any errors find during transmission.

The results were then documented by hand.

esa DOCUMENT

document title/ titre du document

DESIGN FLOW GUIDE FOR AT40KEL-DK

APPENDIX II

prepared by/ *préparé par* Hakan Helzenius

reference/ *réference*
issue/ *édition* 1
revision/ *révision* 0
date of issue/ *date d'édition*
status/ *état* Draft
Document type/ *type de document* Technical Note
Distribution/ *distribution*

European Space Agency
Agence spatiale européenne

ESTEC
Keplerlaan 1 - 2201 AZ Noordwijk - The Netherlands
Tel. (31) 71 5656565 - Fax (31) 71 5656040

Table Of Contents

List of Figures

1 INTRODUCTION

This document is merely a set of experiences collected during the work with the design kit, AT40KEL-DK, from ATMEL. As a first approach only a very small design was tested and when it was functional, more advanced designs were taken through the design flow. This document walks you through the process step by step with inserted comments and reflections. There are several ways of implementing a design into the device, but in this document not all approaches are accounted for. For more information, please see the documentation from ATMEL.

2 DESIGN FLOW GUIDE

This walkthrough is divided into two parts, where the first part describes the purely software oriented elements and the second part the work directly related to the hardware.

2.1 Preparing the Design for Implementation

Included in the development kit is the System Designer 3.0. This software is mainly made for the AT94 series and cannot be used for the AT40KEL FPGA alone. System Designer lacks the right libraries to work with AT40KEL, the programs included in System Designer must instead be used standalone. The software included are Modelsim, Leonardo Spectrum and IDS Figaro. The design flow that is described below is illustrated in figure 2-1 for a better understanding. The process behind most of the boxes in the figure will be described in the following chapters.

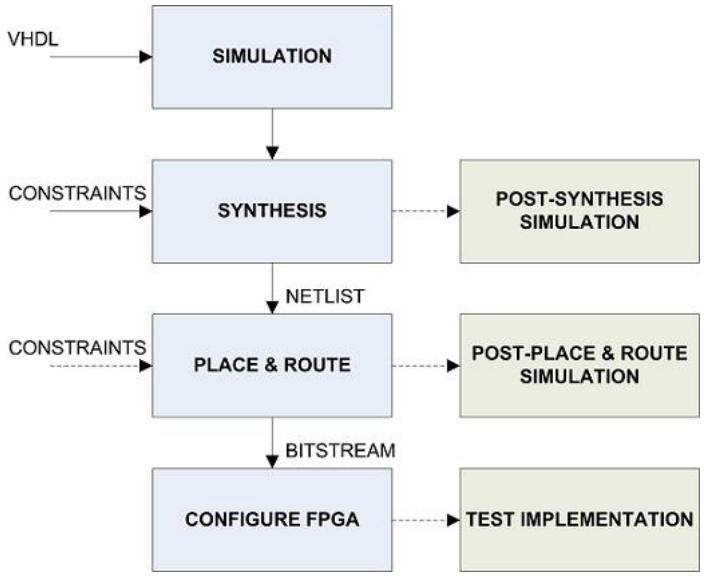

Figure 2-1. Design Flow

2.1.1 PRE-LAYOUT SIMULATION

First of all a simulation of the code must be done, this can be very easy for a simple design but more complicated for larger designs. In this case Modelsim was used but not the Modelsim provided from System Designer 3.0, instead a full license for the windows based Modelsim SE 6.1b was used.

In the pre-layout simulation the design is tested usually with a test bench that send test signals to the design while the result is observed, even though it is possible to force signals into the design without a test bench. For the simple designs the work is straight forward and there may not be a need for many test runs, but with the more advanced designs macros were used to speed up the process. The macros automated much of the work associated with performing a new test run. With the SpaceWire node design from University of Dundee macros for various purposes were included. Minor changes were made to some of these macros for it to work with this software version, therefore it is recommend to use these modified macros when working with the SpaceWire node.

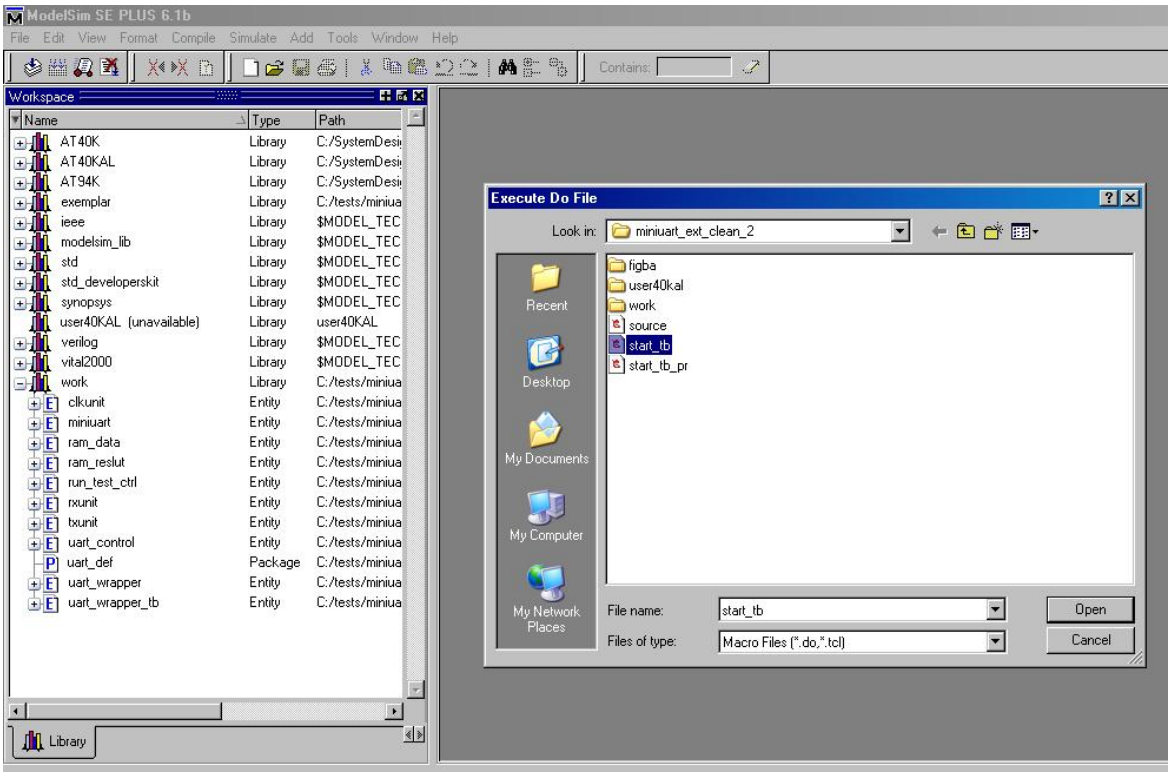

Figure 2-2. Modelsim execute Macros

For the test interface design there were no already made macros but to ease the process small macros were created by coping command lines into DO-files. These macros did usually not include the compilation step since a file path had do be defined, which would limit reusability and some of the time savings would be lost. As a result the main purpose of the macro at this level was to start simulation, add signals to the wave-window and run the simulation for a certain time. The macros mostly used were at first "source" and later on "start_tb". "Source" was a test bench in itself for very small initial tests. Every signal was hard coded into the macro, which pretty soon became too complicated so a more advanced solution had to be created. A VHDL-coded test bench was produced and the "start_tb"-macro to start the simulation using the test bench. In the new test bench whole series of bytes could be defined and sent, which allowed for a much easier testing.

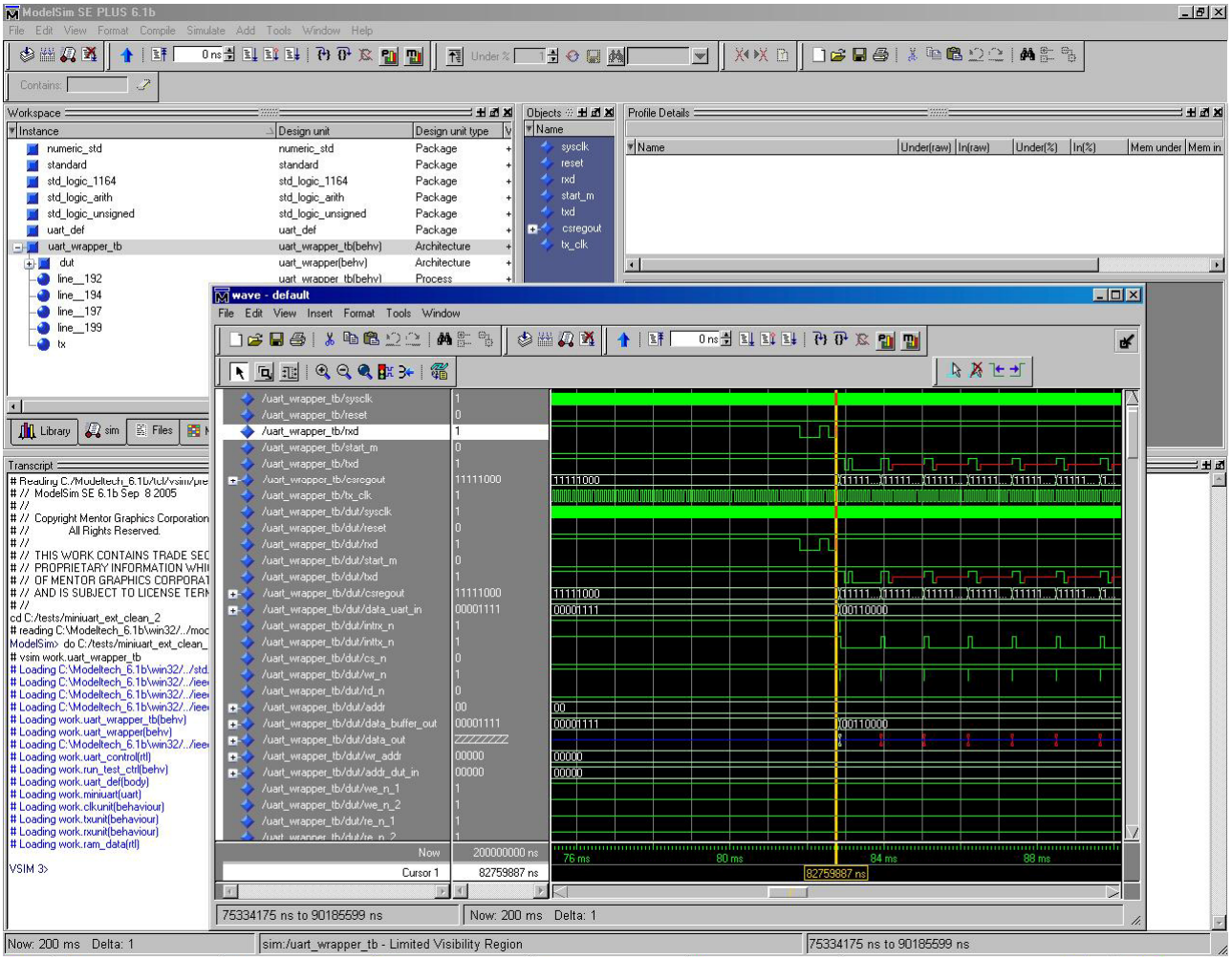

Figure 2-3. Modelsim in the wave window

2.1.2 SYNTHESIS

The tool used for synthesis was Leonardo Spectrum for Atmel version 2002c.37_OEM _Atmel, which was provided by the evaluation kit. The synthesis can be done on many levels, for example a simple design will not require many constraints while a more advanced design may need many complicated constraints. This is of course also a question about how important optimisation is, especially for timings. This project mainly used relatively simple constraints, so for a deeper knowledge in working with constraints please read Leonardo Spectrum documentation.

The "quick setup" option was used for the small test design that was used in the beginning, but the "quick setup" could also be used for first time tries with a more advanced design. Later on, the "advanced setup" was used but constraint wise more complex constraints than in the "quick setup" was rarely used. The "advanced setup" was used mainly to make changes on the output file or to keep the hierarchy after synthesis.

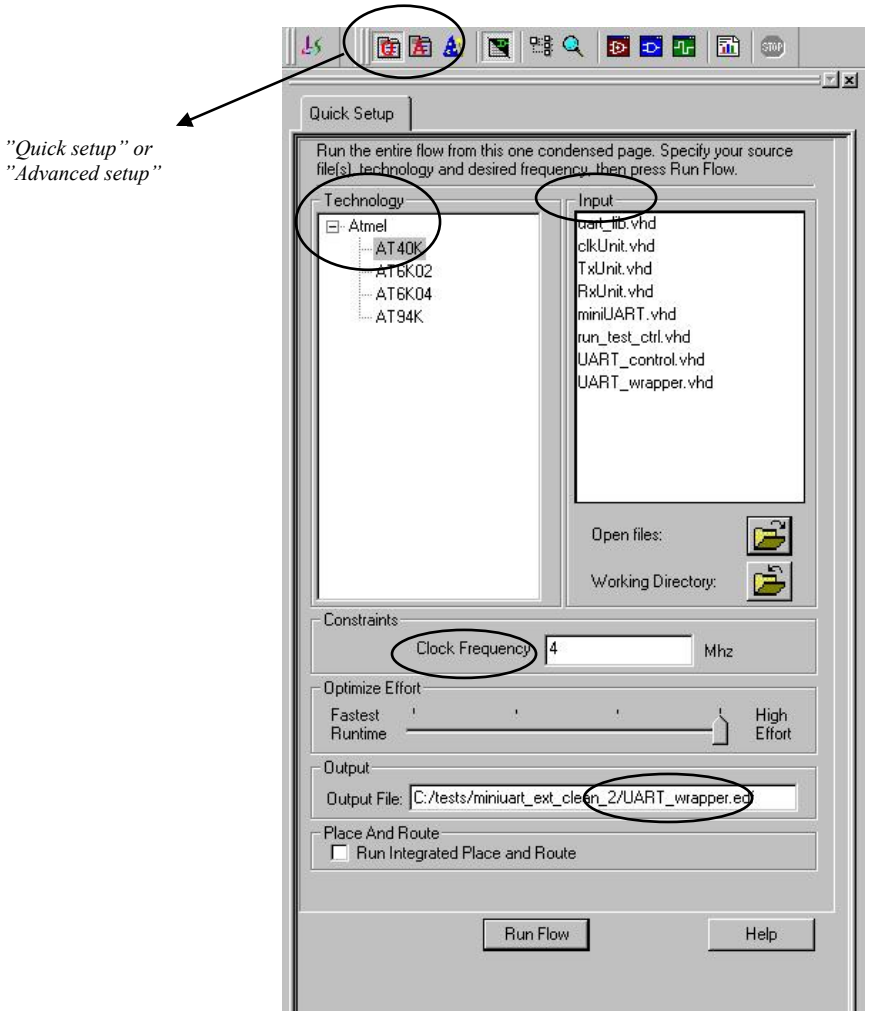

Figure 2-4. Leonardo, Quick Setup

In the “Quick setup” you first choose technology. The figure above shows the right technology, AT40K, as highlighted. At the time there is no special library for AT40KAL or AT40KEL, but it is recommended to check for updates in this area. For instance, with this technology the synthesis is optimised for a supply of 5V instead of 3.3V. Through correspondence with the support they claim this problem is solved by the place & route tool that optimise the design for 3.3V but at the same time they indicated that it might come a new patch with the 3.3V settings. This shows one of the reasons it could be good to be on the look out for a newer patch/library.

After choosing technology it is time to import the input files. You do that by clicking on “open files”. It is important to remember that the order of the input files are vital, where the highest hierarchy file has to be last in the list while the lowest hierarchy file has to be first. Another issue that easily can create mistakes is the “working directory”, which can be set by clicking on the button underneath the “open files” button or when you start a project. The problem is that there is a disturbing delay on when the working directory becomes active. It is very common that it is the old working directory that is active when you want to import files, so if you are not careful you can import an old version of the file since you still are in the old directory. This is also a problem when you start a new project so be always careful so that you import the right files.

A simple constraint can be set here and that is the speed of the system clock. It is only to put in the required clock speed in MHz in the empty box. If your design has high demands on clock speed you probably also need more advanced constraints but it is also good to put the constraint a bit higher since the design most probably will lose in performance during place and route. This loss can be significant when you are working with an FPGA and it was not uncommon to go from a speed of around 30 MHz in synthesis to around 10MHz in place and route.

There is also the option on how high the level of optimisation should be. The highest level was almost always used and the lower level was only used for testing and error solving. There were never any significant differences that could be seen between the levels, though a closer examination was never done.

In the output file option a name and place for the output file can be chosen. The option is preset to an edf-file with the same name as the highest hierarchy input file. This file can later on be exported to the place & route tool. To have more option for the output file was one of the major reasons why the advanced options were used in the later design process. As a last thing it is simply to push the run flow button and read the result in the window to the left of the setup window.

In the “advanced setup” you go through almost the same steps as for “quick setup” but instead of having only a small portion of a window, the option get at least a window for their own. In most of the cases the option has several windows with different sub options. The options that have their own window are Technology, Input, Constraints, Optimize and Output.

Technology is chosen in the same way as in the "quick setup" and there are not many other options. As you see in the bottom of the window it is also more alternatives than in the first window, but neither of the other options were used in the Technology option.

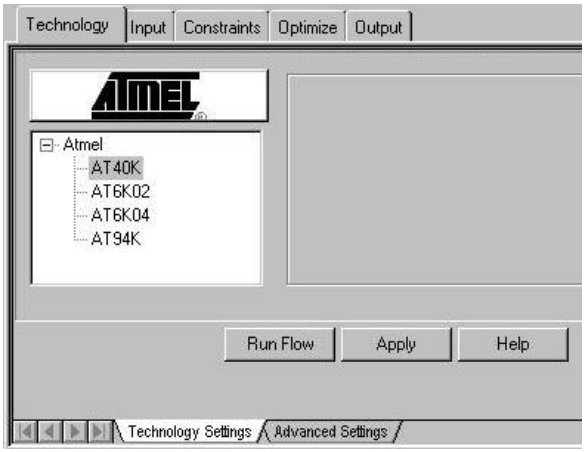

Figure 2-5. Leonardo, Technology Option

"Input" is the second of the options with its own window. Also in this case, only the basic functionality was used. Since only VHD-files were used as inputs many of the sub option were not applicable. You can set the working directory and import files but in addition to that there is also the option to choose encoding style for your state machines for example.

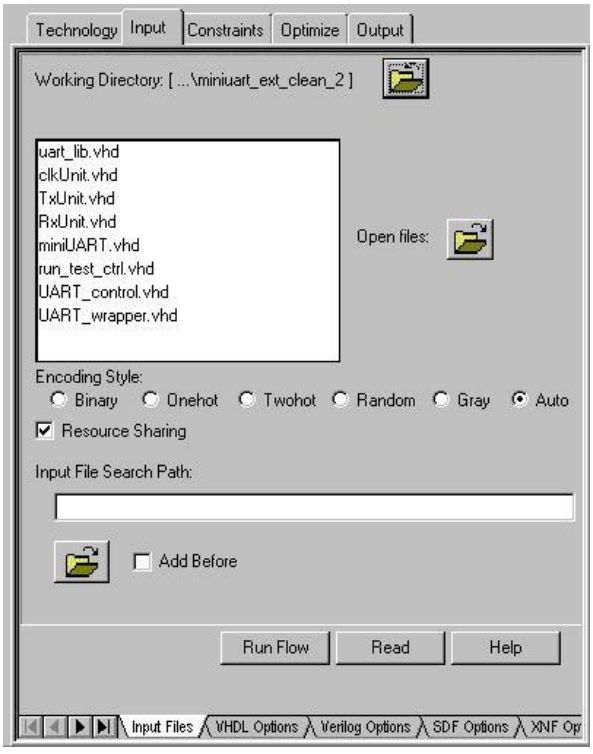

Figure 2-6. Leonardo, Input Option

Constraints can be one of the major functionality differences between the “quick setup” and the “advanced setup” since there is many additional option available here. If you look at some of the sub options on the bottom tabs there are a lot of useful alternatives, like input and output delays, to choose from. Unfortunately there was no time in this project to optimise the design by using more demanding constraints. For more information please see the Leonardo Spectrum documentation.

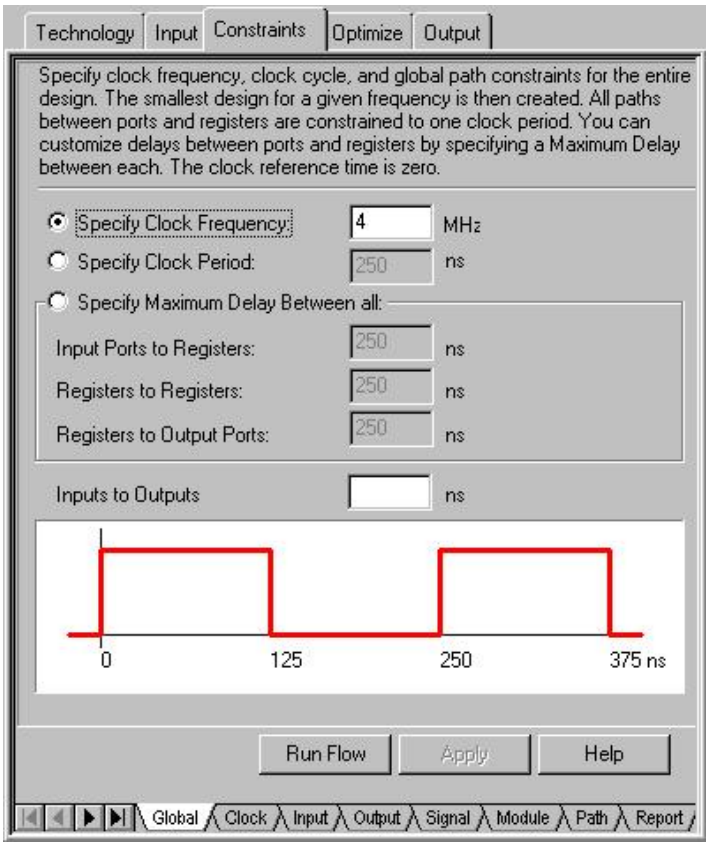

Figure 2-7. Leonardo, Constraints Option

“Optimize” is also an important option, where you can choose which optimisation effort you want or if you want to remap instead. There is also the possibility of choosing different parts of the design to optimise or remap. Even though there are many option here the only thing that has been used more frequently in this project is to preserve the hierarchy for debugging purposes. It is easier to follow the schematic tool when you have the same build up as in your coding. More about this schematic tool will come later on.

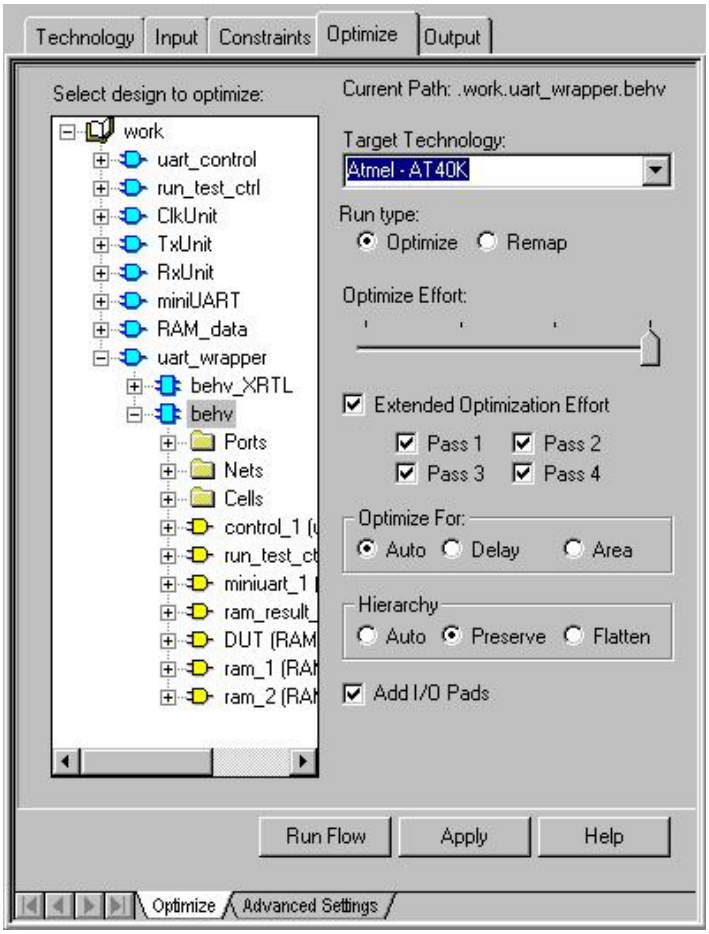

Figure 2-8. Leonardo, Optimize Option

The last one is the "output option" and that is also the option that has been used most in this project. The main reason was to be able to change the output file so that it could be used for post-synthesis simulation. Automatically the settings are set to "auto" which means that the output will be an edf-file, the netlist to use for place & route. If you instead want to do a post- synthesis simulation you most change the output file to a vhd-file, this you do by choosing VHDL as "format". For the post-synthesis simulation there will also be very useful to not "allow writing buses" for a better compatibility with the place & route tool, more information about the convenience of this can be found in the chapter post-synthesis simulation. Anyhow, this setting can be switched on and off if you go to the furthest right among the sub option and click on "VHDL Out Options". In the new window that comes up you can choose to mark or unmark a box for allowing writing busses of the ports.

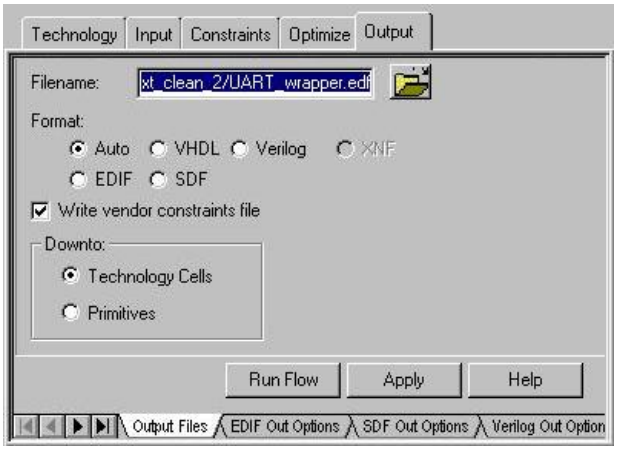

Figure 2-9. Leonardo, Output Option I

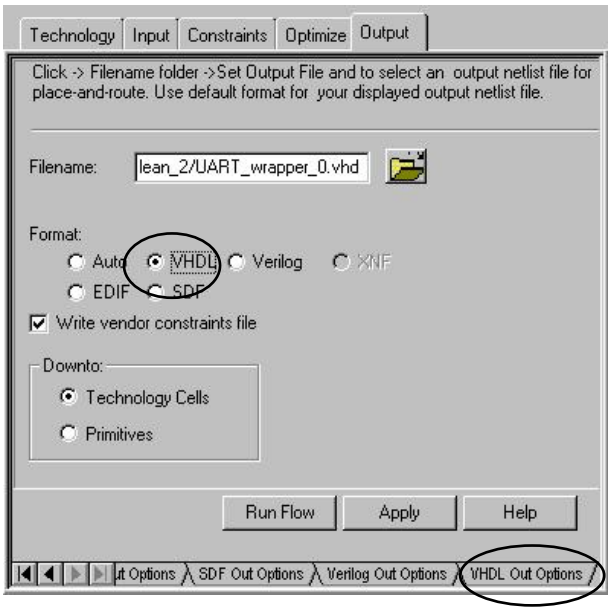

Figure 2-10. Leonardo, Output Option II

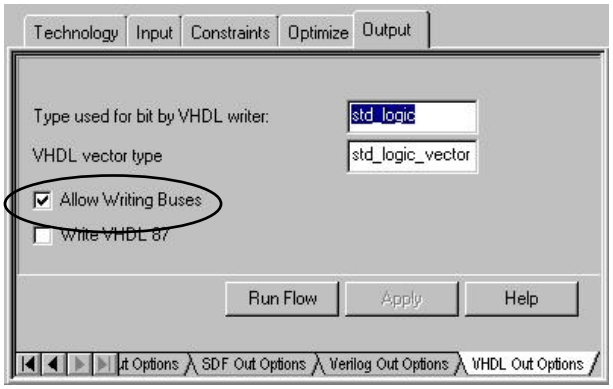

Figure 2-11. Leonardo, Output Option III

After choosing all the required settings, the only remaining thing is to push “Run Flow” and check the result in the information window to the right. It is important to check that all chosen setting have been “applied” and not gone back to default setting, which easily can happen.

The synthesis tool does not only synthesise the design but also provide some other helpful functions like the schematic tool. The schematic tool is a way of getting your design visualised as boxes and connections. This is a convenient tool for looking for errors after synthesis and see if there has been any changes made, especially if the hierarchy is not preserved. You can see two different schematics and figure 2-12 displays an example of a Technology schematic. The black oval shows where you choose to display a schematic.

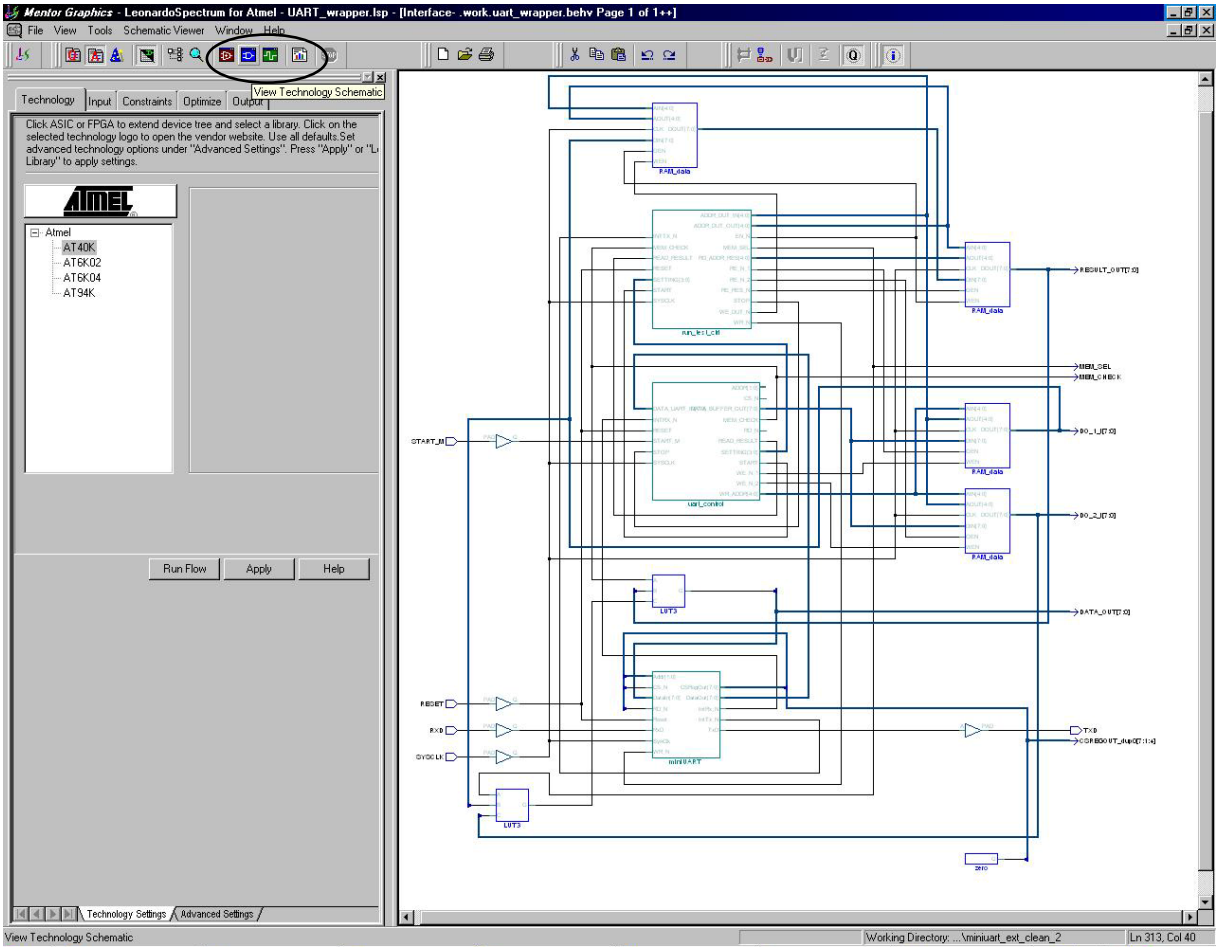

Figure 2-12. Leonardo, Schematic

2.1.3 POST-SYNTHESIS SIMULATION

A post-synthesis simulation is an optional way of testing the design after synthesis. This is done by generating files that can be run in simulation, usually with the same simulation tool as the pre-synthesis simulation. In this case it was a bit tricky since macros generated by the place & route tool was not fully compatible with the file received from the synthesis tool. Macros are design elements that are already predefined in the most efficient way for place & route, so by using them your design will be more efficient. The macros were always used in this project even though it should be possible to turn the option off. There were even plans to synthesise with macros and without and then compare the result but it was not possible within the timeframe of the project.

If you are using macros in your post-synthesis simulation you need to import them from the place & route tool, IDS Figaro. The file you receive from Leonardo, if you choose a VHDL-file as output (see the previous chapter for description on how), have already instanced all the needed macros, so in theory you only need to compile the file from Leonardo together with the files from IDS. But since there are some compatibility problems you need to do some modifications before it works. First of all it is important to **not** "allow writing buses" when you generate the file from Leonardo, this is vital because the VHDL-files of the macros from IDS do not have buses in there ports. Then you also need to call for the AT40KAL library to have the generated file ready for simulation. Now you need to retrieve the VHDL-model for the generated macros from IDS. If you already have generated macros by hand in the IDS tool the models already exist but if they are automatically generated by Leonardo you need to generate the models in IDS. The simplest way of doing that is by running the first steps in the place & route process. This process is described in the next chapter and you only have to follow it until you have generated the macros. Independent on how you generated the macros you copy the VHDL-files from their current folder to a convenient directory. The files can be found in …/figba/"macro name".vhd and is the only VHDL-file in the folder. After that you need to change the names of all the signals, here you have the choice of to make the changes in the file from Leonardo instead but I strongly recommend making the changes in these files. This is because, as it seems, the macros with the same generated name are exactly the same which gives the possibility of reuse. So a recommendation is to keep track of all your already changed macros and you can save plenty of time in the end. There is also the possibility to write a script that will do the name change and other modifications but no such script was created during this project. Now all the files should be ready and you only have to add the AT40K and AT40KAL libraries to Modelsim and compile the files and simulate.

2.1.4 PLACE & ROUTE

To do the place & route you need the netlist (edf-file) from the synthesis tool and how you retrieve it is more closely described in the synthesis chapter. This is a process involving several steps and in many of them changes can be made by hand to further optimise the final result. For this you need a very deep understanding of the tool and the design, so in this project there was no time to reach that level and instead the software's optimisation function was trusted. Below you will get a description on how you take a design through the process.

To open the design and import the netlist for the first time you do the following, you push the open button and choose design in the window that pop up.

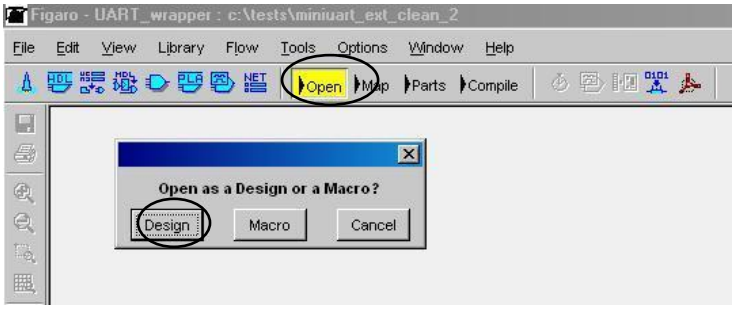

Figure 2-13. IDS, Open Design

Then you can choose to open a new design or an old one. The old ones are listed in the menus while you start a new design on clicking "New Design…".

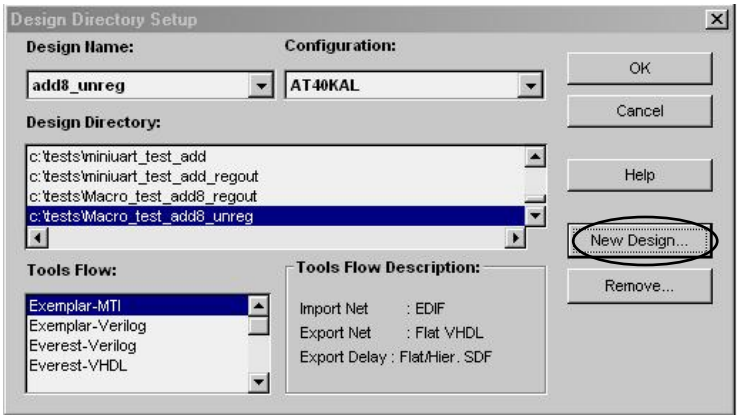

Figure 2-14. IDS, Select Design

If you chose to create a new design a new window will appear, figure 2-14. In that window you can choose a name for your new design, which type of netlist to import and which configuration. In the last choice it is of course important to choose AT40KAL technology.

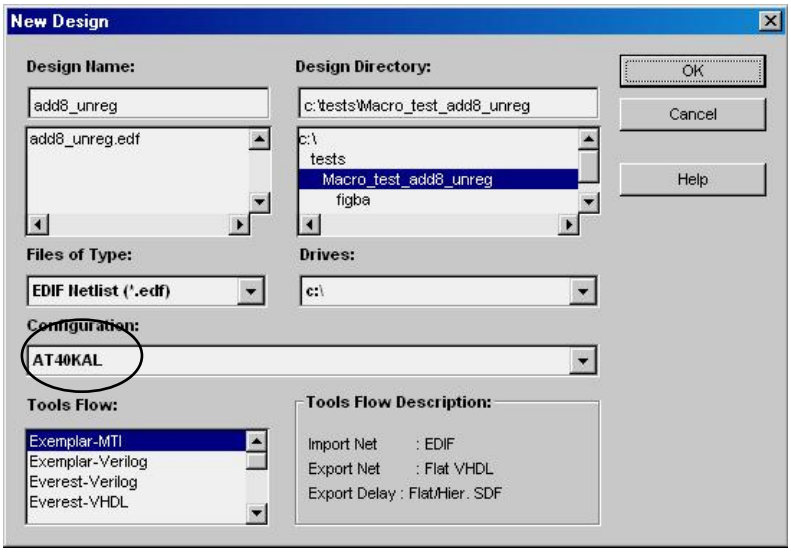

Figure 2-15. IDS, Select Technology

After creating your design file or choosing your old one it is time to import your netlist file. You have two types of file to choose from and if you are working on an old design you may have an fgd-file you want to use but otherwise it is almost always an edf-file you want to import. It is also possible to see in the "Existing Design File" field if there is a file of that type in the specified directory and what the name is of the file. Since you cannot write in the fields yourself in this window it is important to have done the previous setting correctly to avoid problems. For instance it is vital that the edf-file that you want to import is in the same folder as you chose for your design directory.

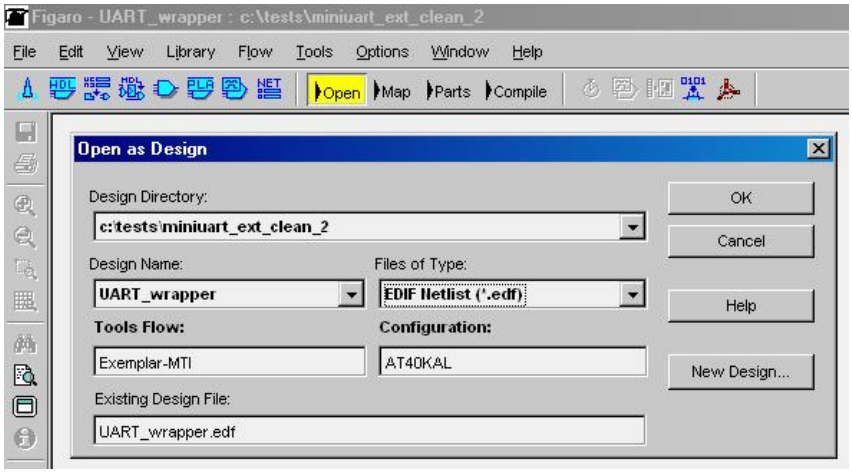

Figure 2-16. IDS, Import EDIF Netlist

When you clicked "OK" and started the design, IDS will read the imported file and if there are automatically generated macros in the design it will start up the macro generator. The window that appears has all the settings prepared by the information from

the synthesis tool. The only thing that needs to be done is to click “Generate”. It can be interesting to see what is generated, so it can be a good idea to look through the macros. Now all the macros are created so if you only wanted them for a post-synthesis simulation, you are done. It has been noticed during this project that if you are reopening and old design file IDS has problems to import the old timings constraint setting and they may have to be set again.

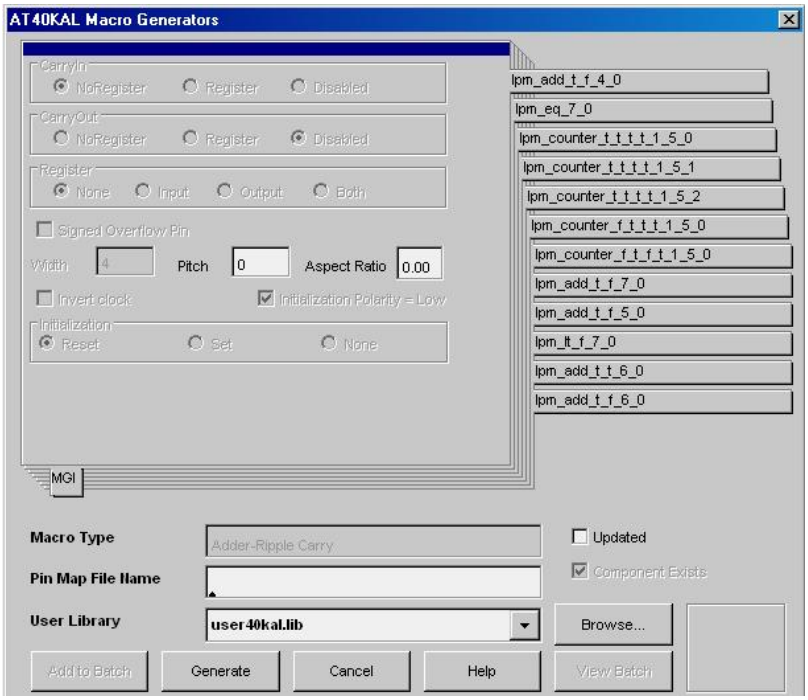

Figure 2-17. IDS, Macro Generator

After that you map the design by clicking on “map” and since there are no settings in this process it is only to wait until it is done. Then it is time to choose a part, so continue and click “part” to get the options in the window in figure 2-18 below. To reduce the number of parts to choose from select “Aerospace” under “Application” and the number to choose from will be reduced. In this project AT40KEL040KW1S was used. In the bottom of the window you can also see how much area and memory this design will take on this particular device.

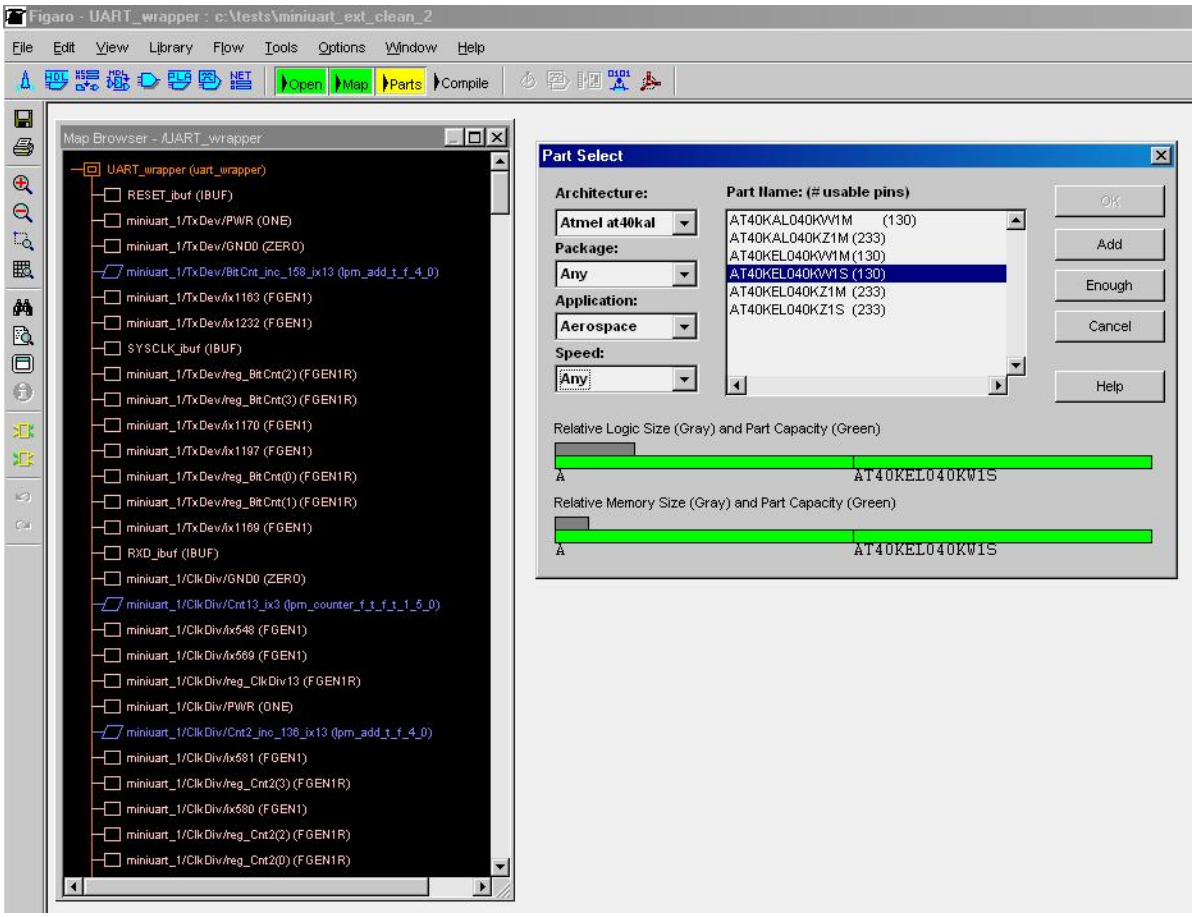

Figure 2-18. IDS, Select Part

After you have selected which part to use it is recommended to assign signals to pins. It is of course a possibility to let IDS choose for you but then you may not get the pins where you want them. To assign pins you go to the “Edit” menu and select “Assign Pin Lock” and a new window appears, where it is possible to lock signals to pins. There is also a possibility to import a pin configuration file, so that you always have the same pins for your design. The easiest way of creating such a file is to save the previous setting. Another good feature is that the pin configuration file is easy to read and work with and even if the file is not a hundred percent correct it still assign the pins that are correct. In the case you have made some small changes among the pins you can still use the old pin configuration file and correct the changes afterwards and save it as a new file, which will save a lot of work. After you have added the part and assigned the pins you also get a graphical view of the device with pins, figure 2-19.

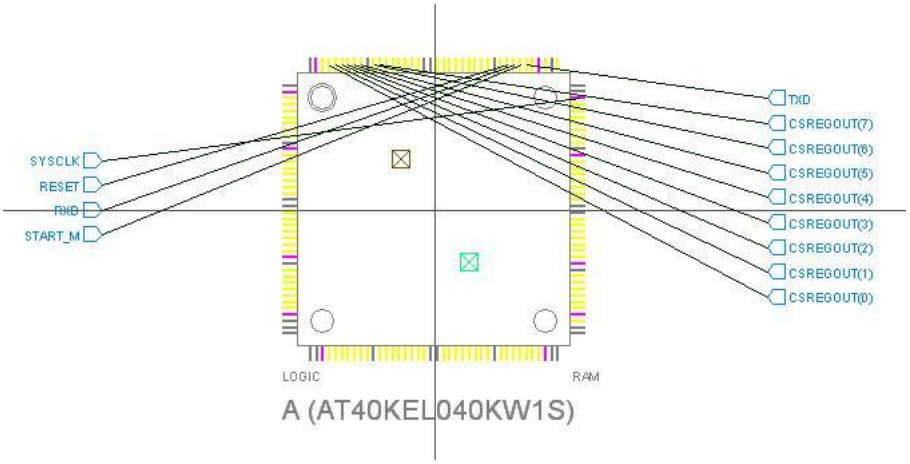

Figure 2-19. IDS, Part Graphical View

The only thing left now in the place & route process is to compile the design. You can do that simply by pushing the compile button but you can also follow the procedure step by step. But before you start compiling the design you have to decide which settings to use. The most important decision is to choose if the place & route should be timing driven or not, a strong recommendation is to use timing driven place & route at least in the later part of the design work. If you choose to use timings driven place & route it is also advisable to define some timings requirements in the "Edit" menu under "timings". Another option that is good to keep in mind is that you can choose to "auto-soften" if there are problems to fit the design into the FPGA. To change these settings and many more you click on "options" in the "options" menu and you will get a window like in figure 2-20. There are so many options here so please refer to the documentation for more guidance.

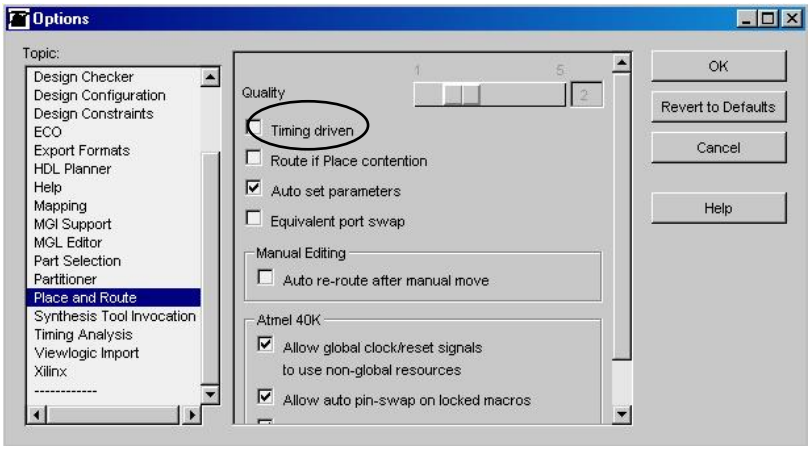

Figure 2-20. IDS, Place & Route Options

Now you can compile the design and if you want to follow it step by step you double click on the drawn out FPGA instead of pushing the compile button. If you do that a window with an "empty" FPGA will appear, see figure 2-21. At top of that window you now have buttons for the "compile flow" in the same way as the design flow was shown in the previous steps.

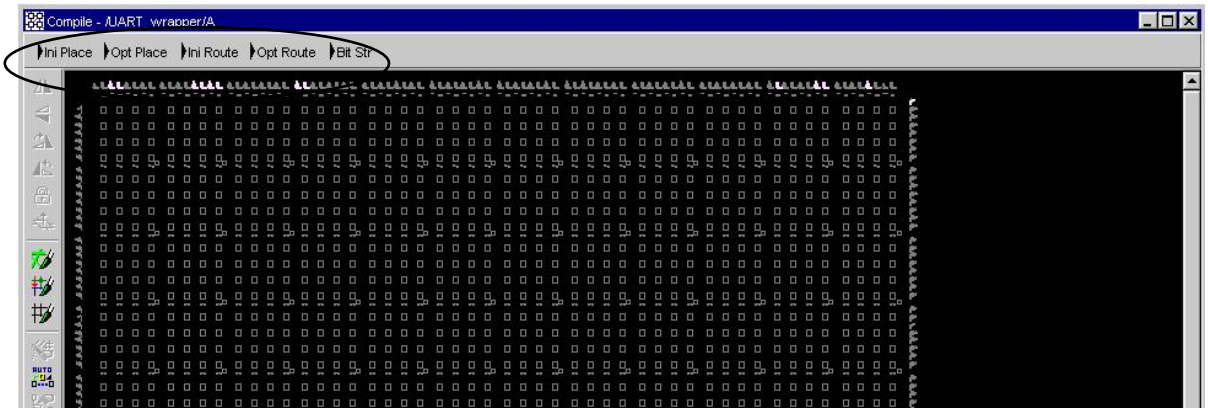

Figure 2-21. IDS, Compile Flow

If you now click on "Ini Place" you will start the initial placement and the whole design will be placed in the FPGA but with no care on placement whatsoever. It is not until you push "Opt Place" you get a placement that is more optimal for timings and area. Figure 2-22 shows a design after optimal placement.

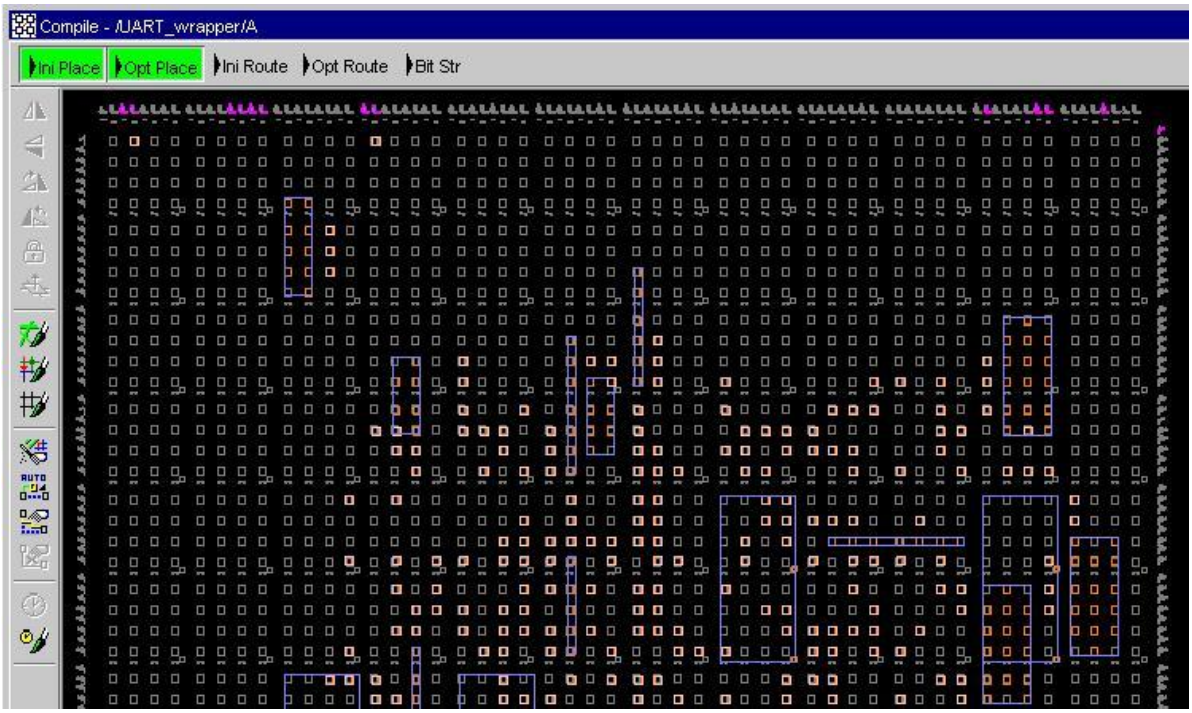

Figure 2-22. IDS, Optimize Placement

When you have done the initial routing you do not see much difference in the FPGA unless you zoom in. Figure 2-23 demonstrates how you can see the routing between the different LUTs and pins. In the figure you can also see that the design has a “contention score” of ten, this means that several connection are made on the same place and if this was the final result the design would not work. In most cases this is not a problem since the “contention score” usually drops to zero after the routing has been optimised.

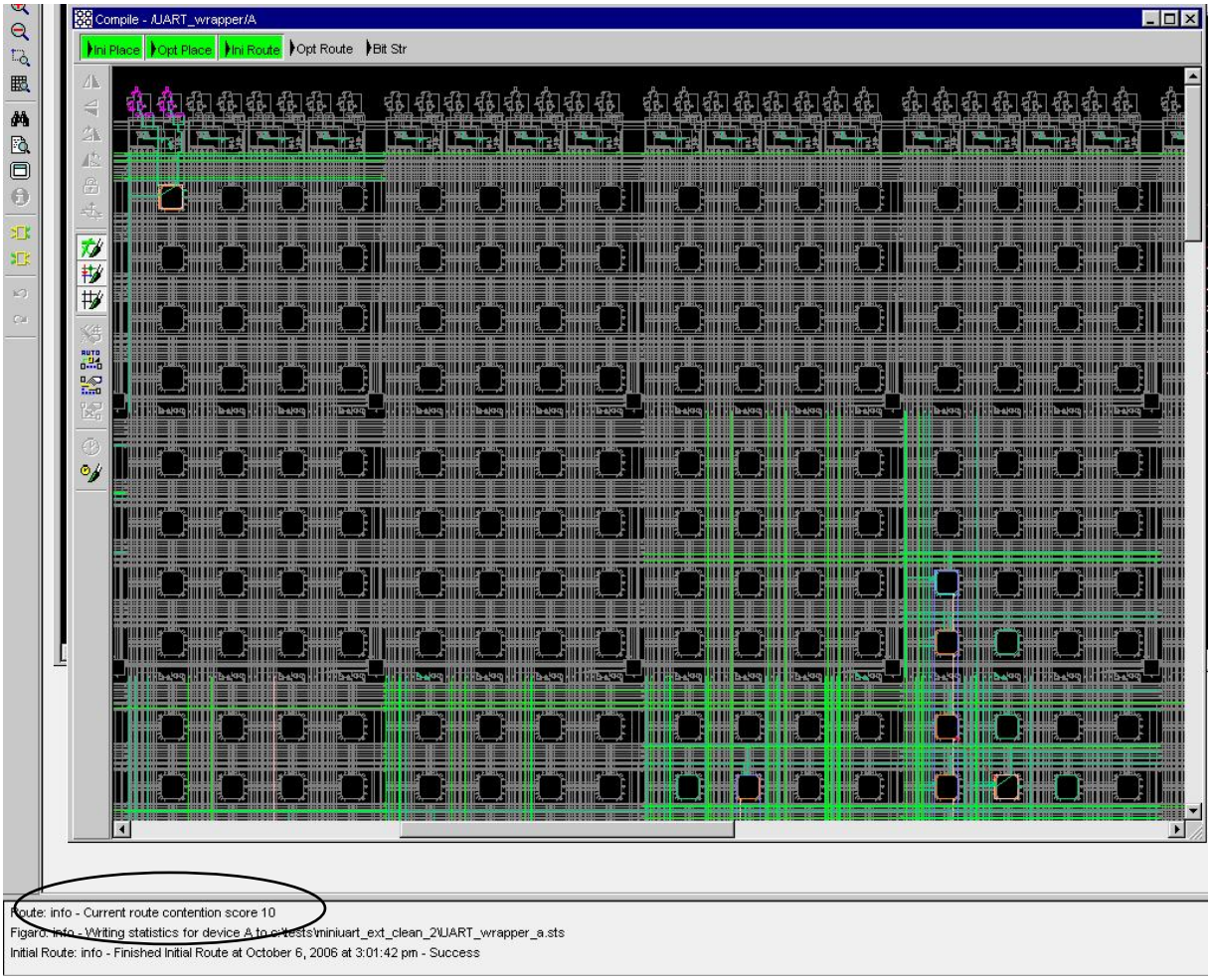

Figure 2-23. IDS, Routing

The last step in the compile process is to create a bit stream. It is during this phase that many of the output files are generated, where the bst-file is the most important. The bit stream file (bst-file) is a file which contains the data that is going to be downloaded into the FPGA. But there is also many other files created and some of them are to be in the post-place & route simulation, which you can read more about in the next chapter. One of the more essential files that are created is the sdf-file with all the timings for the post-place & route simulation.

2.1.5 POST PLACE & ROUTE SIMULATION

This simulation is essential since it is a chance to test the design with the final layout and timings. If this simulation works it is a good chance that the design is functioning in the device as well. To make this simulation work you have to import several files that IDS generated. You need the vhd-files for the generated macros and this time there is no need to do any changes, like you needed to do for the post-synthesis simulation.

To do this simulation you need the libraries, AT40K and AT40KAL, which you used for the synthesis simulation, so if you for some reason do not have them compiled already it is a good idea to do that. The top hierarchy level may not look exactly the same so you better check that it is compatible with your test bench. Hopefully there is not many modifications that needs to be done but it can be good to know that there is also a framework for a test bench generated at the same time the other files are created. In this test bench you only have to add the stimuli but that can be a relatively tedious work if there is not a very simple test bench.

When you have the files and everything ready for simulation there is the possibility to do a simulation without the timings to confirm the functionality, however a simulation with the timings are strongly recommended. To do a simulation with timings you need to import the sdf-file you received from IDS. Working in Modelsim, you do that either in the command line or when you setup the simulation. To do the setup you go to “simulate” menu and choose “start simulation” and in the window that appears click on SDF on the tabs at the top.

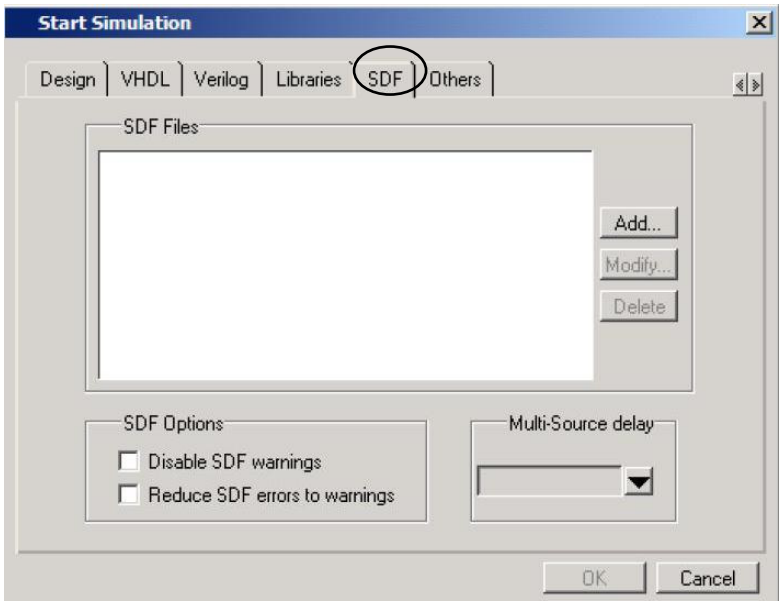

Figure 2-24. Modelsim, Import SDF-file

In this window you can push “add” and you will get a new window where you can choose which sdf-file to import and to which region you want to apply it. You can browse for the sdf-file but for specifying the region you need to write which entity to look in and which instantiation to apply it on. If you are using the generated test bench named “post_test_bench” and want to apply the sdf-file on instance “inst_uart_wrapper”, you simply write it like this “post_test_bench/inst_uart_wrapper”.

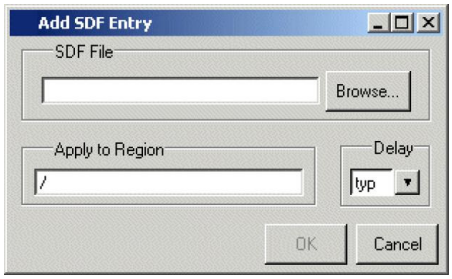

Figure 2-25. Modelsim, Apply the SDF-file

After all the sdf-files have been applied, usally one, the only thing left is to run the simulation as you normally do. In this project the command line was copied into a do-file, “start_tb_pr”, together with some other settings to do the preparation work more straight forward.

2.2 Implementing the design

You can divide the implementation into two parts, the first part is to prepare the equipment and the settings, and the second part is to load the design into the FPGA itself.

2.2.1 EQUIPMENT AND PREPARATION

Most of the equipment that is needed is provided by the development kit but there a few external things that is necessary, like a computer and a power supply. In the kit you can find a parallel cable, to connect the motherboard to a computer, and a mother- and daughterboard with the FPGA already mounted. There is also a CD with the necessary software included in the kit. Together with the power supply that is the only things needed for implementing the design, but then equipment to examine if the design is working properly is recommended. In this project Logic Analysers, a Pattern Generator and a UART interface circuit were used.

The first thing you need to do is to install the CPS-tool, the software that downloads the data to the FPGA. The tool is most probably on your computer already since it is installed together with the other programs on the System Designer CD. You also connect the parallel cable to the board and to the computer, as well as connect the power supply to the board. There are two options when it comes down to power supply, one is to feed the board with 3.3V directly to connector J1 and J2 and switch SW3 on the board to "external" power supply. The other option is to feed the board through connector P1 with 9V and have the board switched to "internal" instead. The latter was chosen for this project as it was easier. When it comes to power it is also very important to have switch SW2 to 3.3V so that the FPGA does not risk to take any damage. LED L1 indicates if the power is on or off, with the board used in this project you had to be extra careful to keep an eye on the LED since the board had a loose connector.

Some other settings that must be set before you start are the switches for which mode you are using to download the configuration to the FPGA. More information about the different modes can be found in the document "AT40K Series Configuration". This project used mode 0, which also is the mode best described on the board itself. To set the board to mode 0 you put all the four switches (CS, M2, M1, M0) to ground,"0". With this mode you first download the configuration to an EEPROM and then to the FPGA from the memory, this will be more closely described in the next chapter.

2.2.2 DOWNLOAD CONFIGURATION

To download the configuration you set the switches, SW4 and SW5, to download to the EEPROM. SW4 should be set to “down” and SW5 to “up”. After that you start up the CPS tool on your computer and set the right configurations, the family should be AT40K and the device AT17LV010(A) (1M). It can be good to check if this is the component name for your EEPROM, you can find more information in the documentation. In addition you choose your bst-file as input file and if you want to change the location of the output file. If you have the same name and location for the output file it will be overwritten, that is good to know if you want to go back and check manually if the transfer was correct.

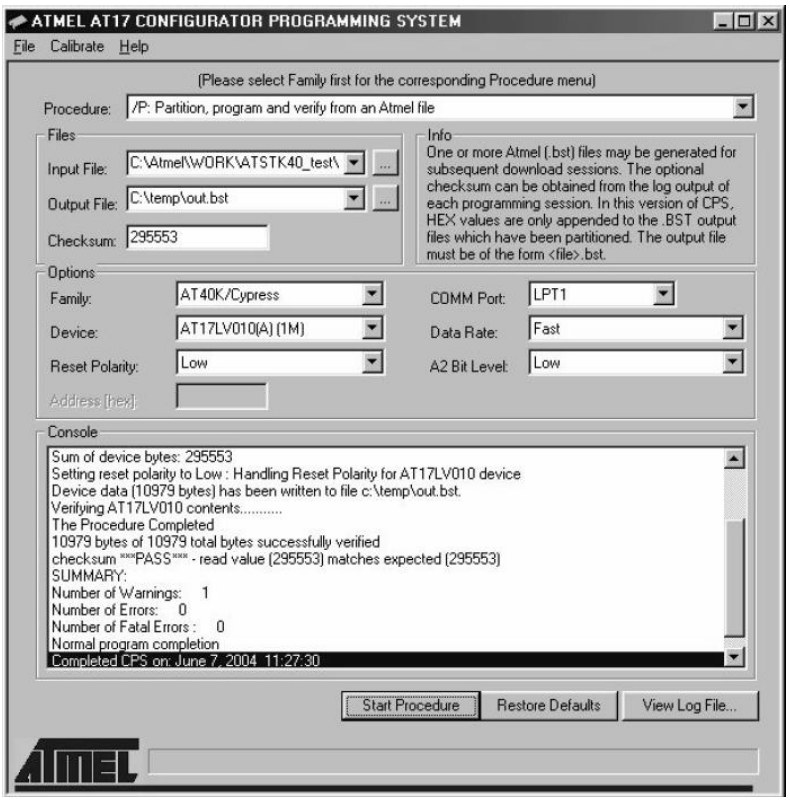

Figure 2-26. CPS

When everything is set in CPS you power on the board with SW1 and then choose “start procedure” in CPS. You will relatively fast get a log in CPS telling you if the transfer was a success or not. If the transfer was successful it is time to read from the EEPROM to the FPGA. If you want to be extra careful you can switch off the board before you change the following switches. Switch SW4 and SW5 to the “Boot from configurator” mode described on the board, SW4 is set to “up” and SW5 to “down”. Power on the board if it was off and now the design is implemented into the FPGA. Control that the clock goes to the right pin, it is changed by a jumper if you use the board oscillator. Connect your test equipment to the right pins and then it is time to start testing the design.

3 CLOSING REMARK

During this process several obstacles were found, most of them could be worked around and they can be read about in this document. Experiences and feed back have also been gathered to be presented as results in the project report. Some of the problems can be better described there and there are also issues not directly connected to the design flow presented in that report. It is therefore recommended to also read the project report.